INTERNET-DRAFT
draft-staniford-cidf-data-formats-00.txt
Expires September 18th, 1998

The Common Intrusion Detection Framework - Data Formats

Stuart Staniford-Chen, UC Davis
Brian Tung, ISI
Phil Porras, SRI
Cliff Kahn, The Open Group
Dan Schnackenberg, Boeing Corp.
Rich Feiertag, Trusted Information Systems.
Maureen Stillman, Odyssey Research Associates

Status Of This Memo

Abstract

This document defines portions of the Common Intrusion Detection Framework (CIDF), specifically the data formats used. CIDF is designed to allow intrusion detection systems (IDS) to interoperate with one another.

Two layered formats are defined here: Gidos, which are a high-level data structure intended to allow IDS systems to exchange messages describing the state of the world, events occurring, and recommended actions with somewhat standardized semantics. Gidos can be encoded in CIDF messages, the format for which is also defined here.

Contents

0.1: Introduction to CIDF

The goal of the Common Intrusion Detection Framework is a set of specifications which allow

* different intrusion detection systems to inter-operate and share information as richly as possible,
* components of intrusion detection systems to be easily re-used in contexts different from those they were designed for.

The CIDF working group came together originally in January 1997 at the behest of Teresa Lunt at DARPA in order to develop standards to accomplish the goals outlined in the previous section. She was particularly concerned that the various intrusion detection efforts she was funding be usable and reusable together and have lasting value to customers of intrusion detection systems.

During the life of the effort, it became clear that this was of wider value than just to DARPA contractors, and the group was broadened to include representatives from a number of government, commercial, and academic organizations. After the first few months, membership in the

CIDF working group was open to any individuals or organizations that wished to contribute. No cost was involved (except to defray meeting expenses).

Major decisions were made at regular (every few months) meetings of the working group. Those decisions were made by rough consensus of all attendees. That is, the meeting facilitator attempted to reach consensus, but in situations where only one or two individuals were protesting a decision, they were overruled in the interest of efficiency. No decisions were taken in the face of opposition from a sizeable minority, rather the issue was tabled for further consideration. Meetings were fun and the working group had a good time doing this (well, most of them, anyway).

In between meetings, most of the writing was done by small subgroups or individuals. Their text was brought back for approval/changes at meetings. Discussions were also carried on in the working group mailing list, but few decisions were made that way.

The CIDF working group is now seeking to become an IETF working group.

0.2: Organization of this document

This section describes the organization of this internet draft on CIDF data formats.

CIDF consists of the following things:

1) A set of architectural conventions for how different parts of intrusion detection systems can be modeled as CIDF components.

2) A way to represent gidos (generalized intrusion detection objects). Gidos can
   * describe events that have happened in the systems mo by an IDS,
   * instruct an IDS to carry out some action
   * query an IDS as to what has happened.
   * describe an IDS component.

3) A way to encode gidos into streams of bytes suitable for transmission over a network or storage in a file.

4) Protocols for CIDF components to find each other over a network and exchange gidos.

5) Application Programming Interfaces to re-use CIDF

components.

This internet draft is mainly concerned with the Gido data structure, (appearing in Chapter 2), how gidos get encoded on the wire (appearing in Chapter 3), and the message formats (in chapter 4). However, as an orientation, the architectural view is covered in Chapter 1.

APIs and mechanisms for location of components are not discussed in this draft.

0.2.1: Format

This document complies with the requirements for RFC 1543, the format for ASCII Internet RFCs. In summary, this means that lines are at most **72 characters long and that they are terminated with a carriage-return,** line-feed pair. Pages are at most 58 lines long and are terminated with a form-feed character. Paragraphs are single spaced and are separated by blank lines.

Lines in the text beginning with "#" denote editorial comments which should be removed before the final version.

The document is also divided into sections which are further divided into subsections, subsubsections, and so on. The numbering convention is as "3.4.1", which describes the first subsubsection of the fourth subsection of the third section. Appendices are lettered, and so an Appendix subsection might be B.4.2.

```
================================================================================
================================================================================
=
=                          1: CIDF Architecture
=
================================================================================
================================================================================
=
================================================================================
= 1.1: Introduction
================================================================================
```

This section introduces the architectural framework that CIDF assumes will structure an intrusion detection system. This scheme is basically a framework around which interfaces and the communication protocols are organized. It is not mandated that CIDF-conformant intrusion detection systems must be organized in exactly this way. But they must support interfaces that are so organized.

Section 1.2 introduces the various different kinds of components that CIDF believes are needed in IDS systems. Section 1.3 covers the

communication layering scheme, and section 1.4 discusses how components are named and located.

1.2: CIDF Functional Decomposition

All CIDF components deal in *gidos* (generalized intrusion detection objects) which are represented via a standard common format. Gidos are data that is moved around in the intrusion detection system. Gidos can represent events that occurred in the system, analysis of those events, prescriptions to be carried out, or queries about events.

CIDF defines four interfaces that CIDF components may implement:

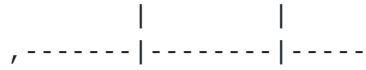
	Push-style	Pull-style
Producer	Produces gidos when it wants to, typically in response to events.	Produces gidos when queried.
Consumer	Mates with push-style producer.	Mates with pull-style producer.

Each of these interfaces takes two forms: a callable form, which permits reuse of the component, and a protocol form, which permits the component to interoperate with other CIDF components.

CIDF defines several types of preferred components:

* Event generators
* Analyzers
* Databases
* Response units

Figure 1.1 presents a schematic view of these components in a hypothetical intrusion detection system. The solid boxes labeled E1, E2, A1, A2, D, etc represent the various components of some hypothetical intrusion detection system. It is convenient to think of these as objects in the object-oriented programming sense (this does not dictate an implementation in an object-oriented language or framework).



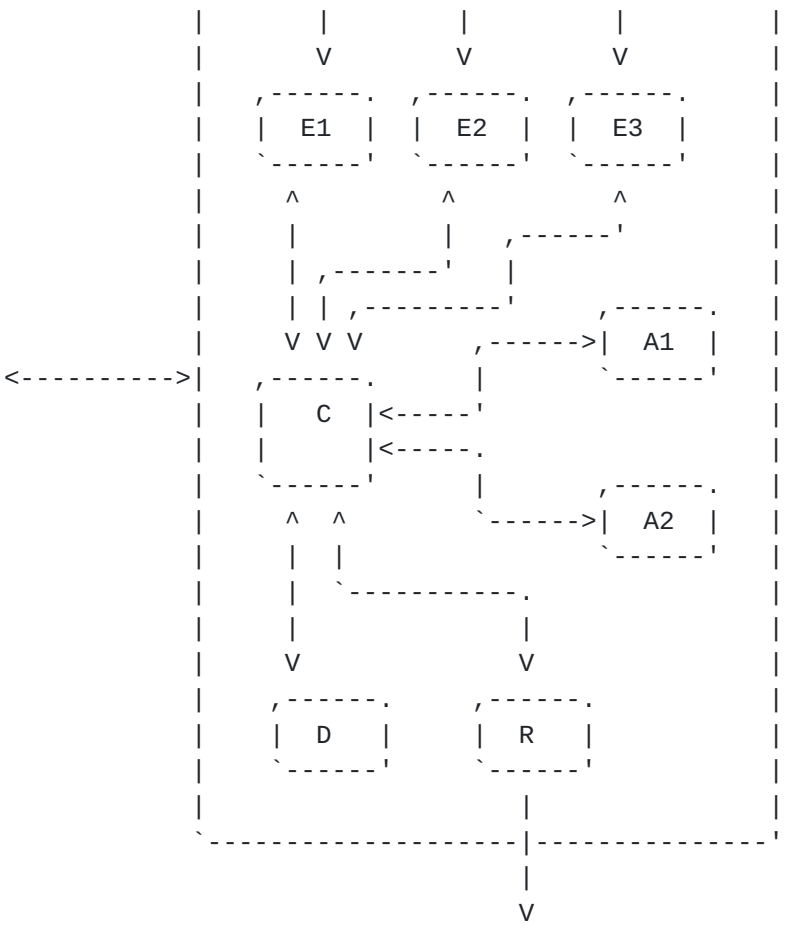


Figure 1.1: Types of CIDF components

Whether the individual components are separate processes or images, or merely conceptually separate parts of the code in a single image is not specified - both possibilities are covered by the CIDF specification.

CIDF allows for components to be aggregated together to masquerade as a single component. In other words, a large number of (possibly distributed) components can be tied together and present themselves to the outside world through a single CIDF interface.

```
#########################################################################
#
# Stuart comment:
# It is not clear at present how this last requirement is to
# be achieved.
#
#########################################################################
```

1.2.1 Configuration and Directory Service

The box labelled C in Figure 1.1 represents the configuration and directory services that tie components together via their standard CIDF interfaces. A component initiating communication may avoid using these services if it knows how to address its target directly, or uses non-CIDF means to do so. Otherwise, these services allow a component either to look up its target explicitly or to derive its communication "partners" by looking up "gido classes".

Gido classes specify types of data that may be exchanged between components. Components that wish to receive certain kinds of gidos describe what they want; components producing event records describe what it is they produce. The directory service, augmented by intelligence local to each component, then takes care of associating GIDO producers with appropriate GIDO consumers. In this mode of use, components are thus relieved of the burden of identifying or locating their partners in the intrusion-detection system.

This service is not discussed further
in this Internet draft.

1.2.2 Event Generators

The boxes labelled Ei in Figure 1.1 are event generators. Their role is to obtain events from the larger computational environment outside the intrusion detection system (symbolized by the arrows coming from outside the large box), and provide them in the CIDF standard gido format to the rest of the system. For example, event generators might be simple filters that take C2 audit trails and convert them into the standard format. Another event generator may passively monitor a network and generate events based on the traffic thereon. A third might be application code in an SQL database program which generates events describing database transactions.

It seems that event generators are likely to be reusable in that CIDF has a standard data format, and so converting features of typical computational environments into that format will be a task that many groups will need to perform. Hence, it is useful to specify a preferred way to configure and use event generators.

Preferred event generators implement the push-style producer interface. They create only gidos describing raw events, not gidos describing analyses or prescriptions.

Preferred event generators provide events as soon as they occur (with the possible exception of transport queuing). Storage of events is handled in gido databases.

1.2.2 Event Analyzers

Analyzers are labeled by Ai in Figure 1.1. They are the components we typically think of in the intrusion detection context. They obtain

gidos from other components, analyze them, and return new gidos (which hopefully represent some kind of synthesis or summary of the input).

Thus for example, an analyzer might be a statistical profiling tool that examines whether events being supplied to it now are statistically unlikely to be from the same time series as events supplied to it in the past. Another example is a signature tool that examines sequences of events looking for particular patterns that represent known misuse of the system. Another example would be a correlator that simply examines events and attempts to determine whether they are causally related to one another, and then puts them together into composite events which can be further analyzed. Simple analyzers might be just filters that throw away events that match certain patterns, or caches that only forward events dissimilar from recently seen events.

A preferred event analyzer implements the push-style consumer interface, whereby it obtains input, and the push-style producer interface, whereby it reports analyses. The gidos it produces are analysis results, not raw events nor prescriptions.

Again, preferred gido analyzers immediately pass through gidos (with the exception of some processing delay). No provision is made for storage of gidos by analyzers.

1.2.3 Event Databases

Databases are labeled by Di in Figure 1.1. These components exist simply to give persistence to CIDF gidos where that is necessary. The interfaces allow other components to pass gidos to the database, and to query the database for gidos that it is holding. Databases are not expected to change or process the gidos in any way (or at least to maintain the illusion that they don't).

A preferred gido database implements the push-style consumer interface, whereby it receives any sort of gido,
and the pull-style producer interface, whereby it responds
to queries.

It is not assumed that the database is a complex application (such as a relational database). It may simply be a file.

1.2.4 Response Units

Response units are the soldier ants of the CIDF ant-heap. They carry out prescriptions - gidos that instruct them to act on behalf of other CIDF components. This is where functionality such as killing processes, resetting connections, etc. would reside. Response units are not expected to produce output except as acknowledgements.

A preferred response unit implements the push-style consumer interface, whereby it receives prescriptions. It may also implement the push-style producer interface, whereby it reports on its efforts to carry out the prescriptions.

1.2.5 Other Components

Many other useful types of component are compatible with CIDF. For example, a subsystem may record events in a non-CIDF format, but may implement the pull-style producer interface so that CIDF components can query its record of events.

A component may record gidos for archival purposes, thus needing only a push-style consumer interface.

A component may observe the world and do some analysis or filtering before creating gidos. Such a component implements the push-style producer interface.

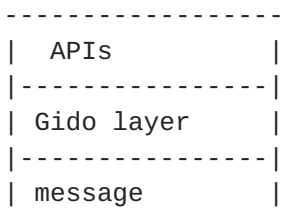
An event analyzer may consult a gido database. The analyzer would need a pull-style consumer interface beside the usual push-style producer and push-style consumer interfaces.

A component may carry out responses, like a response unit, but also produce analyses, like an event analyzer.

1.3: Communication Layers

1.3.1: Background

CIDF supports both interoperability and reusability of components. As such, a component may be communicating with another across the network, or as part of the same executable. In addition, to the extent feasible, CIDF avoids specifying a particular language or choice of network protocols. To support this flexibility, the design is structured in layers. Figure 1.2 shows the layers.



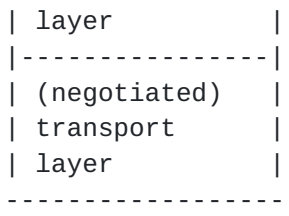


Figure 1.2

1.3.2: API Layer

At the top of figure 1.2 is an API layer indicating code-based interfaces to the layers below. This is not discussed in this draft.

1.3.3: Gido layer

Independent of programming language, network protocols, etc, CIDF defines common formats for intrusion detection data. This data comes in discrete packages called gidos (generalized intrusion detection objects). The organization of the data, its semantics for an IDS component, and a way to encode it in bytes are all defined at this level.

The rationale for this is to separate the issue of how data is organized and what it means (gido layer) from how it is gotten in and out of components (API layer) and moved across networks (message layer). Gidos are discussed in sections 2 and 3 of this document.

1.3.4: Message layer

Gidos must be moved across networks. Certain features of this process must be present for CIDF purposes and may not be provided by underlying transport mechanisms (such as cryptography, CIDF addressing, etc). The CIDF message layer is intended to provide this functionality. This layer is addressed in section 4. Use of this layer is mandated for CIDF components that are to be interoperable across a network.

1.3.5: Transport layer

The figure below illustrates the notion of two independently developed CIDF modules that build to a common interface specification. CIDF supports

For the two modules to communicate, they are required to employ the same transport protocols that will establish the communication channel and handle message passing. The introduction of the transport layer is handled during the integration phase, as module developers negotiate and agree upon a common transport channel. For example, both developers may agree that sockets will be used for this communication session. Other

developers may decide they wish to employ secure RPC for a different session. CIDF provides the flexibility to use different transport mechanisms, and a negotiation mechanism to choose amongst them. The reason for having an independent transport layer below the message layer is that our only requirement is that the components understand the messages. This is independent of the way in which messages are transmitted. Different applications will require different transport mechanisms. All components are required to support a default transport mechanism, namely UDP. This is necessary in order to guarantee that two components can talk at least enough to negotiate about which other transport mechanism they might prefer.

Interoperation Among Independently-Developed
Intrusion-Detection Modules

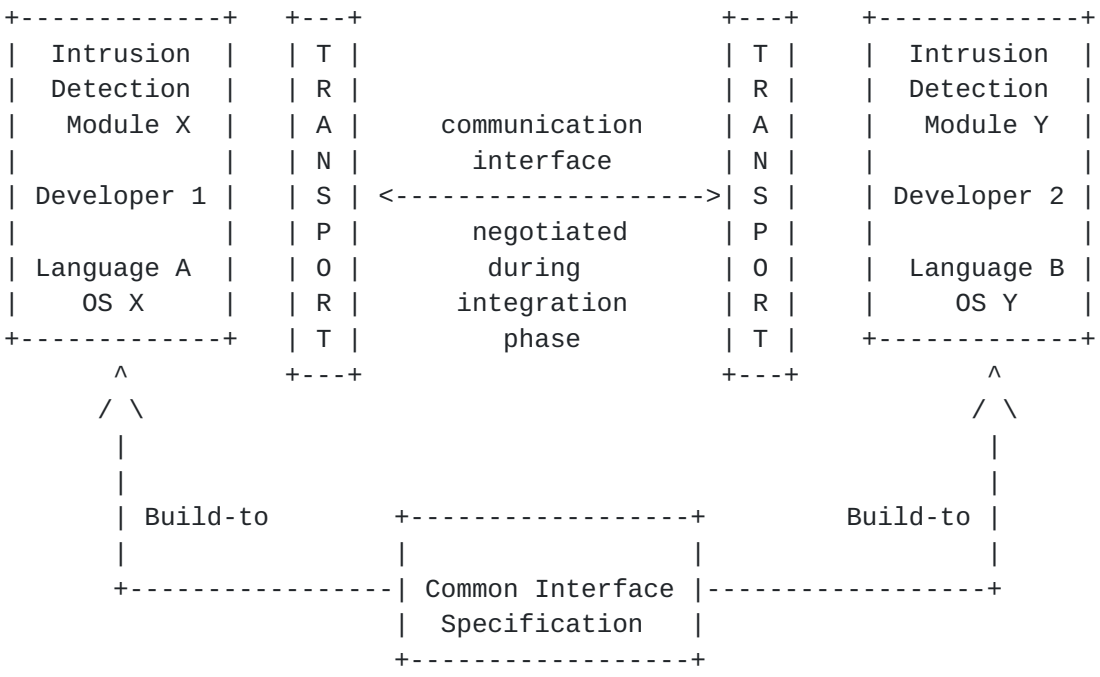


Figure 1.3

= 2.1: Introduction to the gido format

2.1.1: Overview

This chapter specifies a standard gido format for use by CIDF components. These components shall use this standard for disseminating event records, analysis results, and countermeasure directives, to IDS modules. The document both defines the syntactic structure of these messages, and provides a method for defining the semantic content

necessary for interpreting the various data elements embedded within the structure.

2.1.2: Organization

This section is organized as follows. Section 2.2 discusses the requirements for the gido format and the rationale for our choice. Section 2.3 summarizes S-expressions as we define them and use them for gidos. Section 2.4 begins serious discussion of the semantic identifiers we use, and how to put gido-sentences together. Section 2.5 provides some detailed examples. Section 2.6 contains some rules and guidelines for defining new SIDS. Section 2.7 identifies the recommended set of GIDOs (primarily internal status information) that all CIDF-compliant modules should be able to produce. Section 2.8 discusses requirements for gido format negotiation protocols.

Readers will probably wish also to consult Appendices A and B which list all the currently defined data types and SIDS.

2.2: Gido Requirements and Rationale.

Under the CIDF data sharing model, components receive an input stream, use this input to drive their internal analytical processing, and pass the results to other components within an overall intrusion detection architecture. The output of one component may be the input of another component. Therefore, this specification closely coordinates the structures of event records, analysis reports, and countermeasure prescriptions. In many cases, current state information must also be used in order to fully understand the meaning of events, hence this is also encoded in gidos. This adoption of a single standard for both E-, A-, and R-boxes provides significant advantages in the reduction of interface complexity. In addition, this approach provides great flexibility as intrusion-detection objectives move from component analysis, to systems analysis, to system of systems analysis.

However, this relationship between event records and analysis results does not necessarily extend beyond the specification of identical gido structures. Event records, analysis results, and countermeasure prescriptions remain dissimilar in significant ways:

- Event records represent the operational activity of the analysis target, and may be produced in large volumes. Minor losses of event records, while potentially damaging, will not necessarily imply a significant compromise to operational security.

- Analysis results represent significant conclusions derived from an analytical review of an event stream, and should represent a

significant reduction in volume from that of the event stream. Minor losses of analysis results are far more critical to the operation security of the target system than event records.

- o Countermeasure results likewise should be low volume and sensitive to loss.

Thus, while gidos encode events, analysis results, and countermeasure prescriptions identically, other processing layers such as transport may handle them differently. For example, specifications for event transport may derive requirements that emphasize performance (e.g., stateless UDP transmission), while analysis results dissemination protocols may emphasize ensured delivery and accurate reassembly over issues of performance (e.g., TCP transmission). Protocols for event dissemination and analysis results reporting may also handle other issues differently, such as security requirements.

The GIDO structure contains the actual data representing the event record, analysis results, and countermeasure directives produced by their respective CIDF components. The encoding scheme requires the ability to express complex, self-defining data structures, while providing efficient high-volume transmissions of predefined structures. This specification uses S-expressions as the basic payload format.

S-expressions are a self-defining formatting scheme for representing arbitrarily complex data structures. This message encoding specification employs a very simplified form of S-expressions for event record, analysis report, and countermeasure directive representation. One of the motivations for this choice is that S-expressions in general allow for an impressive degree of reasoning and formalism.

The design goals for the gido format are:

- -- generality: Gidos should be capable of representing arbitrarily complex data.
- -- self-defining: Extensions to payload formatting should be semantically defined within the payload itself. Consumers should be able to learn or adjust to alterations in the expected format or comprehend entirely new payload format.
- -- simplicity: The encoding scheme should produce messages that do not force complex parsing logic upon IDS module developers if tha is not necessary in their application. The encoding scheme should b easily understandable and gidos should have a human readable representation.
- -- efficiency: Payload expressions should represent data compactly. The overhead of semantic self-definitions should be removable when predefined messages are transported in bulk.
- -- flexible: Payload expressions must be open to modification and

extensions to new data types, semantic information, and new data structures.

-- independent of call semantics: Payload expression must be supportive of both embedded data (call by value) messages and data independent (call by reference) messages.

2.3 GIDO S-expression format

2.3.1 Preamble

In this section, we define how S-expressions are put together at a low level in CIDF. This is the human readable format; the wire format is defined in terms of this one in section 3.3.

In addition to questions of encoding format, this specification also enumerates a set of CIDF-compliant default primitive data types and semantic-identifiers (SIDs) used when expressing individual payload fields. How SIDS should be combined into S-expressions that form meaningful gidos is discussed in section 2.4

The primitive data types, presented in Appendix A, define the available encoding used for field representation. Semantic-identifiers (SIDs), in Appendix B, provide standard identifiers that gido consumers may use to interpret the various data fields within a payload expression.

2.3.2 S-Expression Grammar

Following is the grammar for CIDF S-expressions in BNF. Terminal symbols are represented in upper case. Literal characters are enclosed in quotes (").

```
<item-list>     ::= <item> | "(" item-list ")" |
     <item-list><item>
<item>     ::= "(" <sid-exp> <data-exp-list> ")" |
     "(" "def" SID <sid-exp> ")"
<sid-exp>     ::= <specifier> |
     "(" <specifier> <sid-exp-list> ")"
<sid-exp-list>  ::= <sid-exp> |
     <sid-exp-list> <sid-exp>
<specifier>     ::= SID | TYPE | NAME
<data-exp-list> ::= <data-exp> |
     <data-exp-list> <data-exp>
<data-exp>     ::= DATA |
     "(" <sid-exp> <data-exp-list> ")"
```

Using this grammar, data fields are coupled with semantic identifiers parenthetically. A SID indicates how its associated data element is

syntactically represented as well as the data element's semantic content. A collection of parenthetical SID/Data tuples can themselves be grouped together in outer parentheses, indicating an explicit *association* of the SID/Data tuples (i.e., they represent attributes of a larger element in the expression). SID grouping is discussed further, with illustrations, in Section 2.3.

A SID is a unique token for a semantic identifier. TYPE is one of the primitive types specified in Appendix A. NAME identifies a named element of a structure. DATA is a data literal.

2.3.3: GIDO S-expression Examples

The following sections illustrate low-level ways of using S-expressions to encode gido data structures. We give these examples for concreteness, but see the next section for more information on how to form gidos.

2.3.3.1: Embedded Semantics and Data Payload Example

This form is used for expressing field-oriented lists of data, where the data is embedded within the message. The format consists of a series of tuples, one tuple per data field. Each tuple consists of a semantic identifier followed by its associated data item:

```
Format: (SID-1 data-exp-1)(SID-2 data-exp-2) . . . (SID-N data-exp-N)
```

In this format, their is a SID with each data item, providing a self-defining message format. A consumer can parse the message for those SIDs it understands and desires to analyze, and discard data fields containing unknown or unwanted SIDs. As discussed in Appendix B, each SID has an associated data type, which completes the self-definition of the message. Thus, by parsing the SID tokens, the consumer knows both how to interpret each data element semantically, and how the data elements are syntactically represented.

2.3.3.2 Pre-defined Constant Payload Format

This form allows for semantics of predefined message structures to be conveyed to consumers once. From that point forward, consumers can receive and interpret raw data structures without the overhead of embedded SIDs. This form is highly efficient for transporting high-volumes of the same message type. This form is also used for enumerating a pre-defined set of CIDF E/A-box messages (see Section 2.5).

A gido producer begin the message exchange by sending the consumer a message definition statement. The "def" defines a new SID that can be used subsequently. SID indicates the semantic identifier being defined.

SIDs are special identifiers in the language. Attempting to define a SID that is already defined is an error. arg-list is a list of dummy arguments that will be matched with the actual arguments in use to evaluate the S-expression. sid-exp-1 defines the SID in terms of SIDs and TYPEs that are already defined. sid-exp-1 may only contain SIDS that have been predefined either because they are included in an appendix to this document or they have been defined in a prior definition.

Format1: (def SID arg-list sid-exp-1)

```
##########################################################################
# Editor's Comment: The event subgroup has not resolved the
# issue of scope for dynamically defined SIDS.
##########################################################################
```


= 2.4 Parts of a GIDO payload

2.4.1 Introduction

A GIDO consists of the GIDO header--which gives information pertaining to the encapsulation of the GIDO, such as its version number, its length, and so forth--and the GIDO payload. In this section, we will describe how SIDs are put together to compose the GIDO payload using S-expressions described in the last section. The Gido header is discussed in section 3.2

A well-formed GIDO payload consists of one or more top-level *sentences*.

Sentences are S-expressions that can be said to "assert" something. A typical sentence might describe the state of a machine at a given time, or it might report that a given event had taken place, or it might also recommend that an action be taken to counter an attack.

A sentence may be composed of other sentences, connected in some way; such a sentence is called a *compound sentence*. A sentence which is not compound is called a *simple sentence*. Broadly speaking, a simple sentence contains a *verb*, which describes what happened, and other S-expressions that describe who verbed what, where, when, and how, and so forth.

In the following sections, we will examine how each of these may be denoted and described, and finally, put together to form a complete sentence.

2.4.2. Verb SIDs

At the heart of a sentence is the *verb*. Normally, we think of verbs as denoting some action (which may sound somewhat event-centric), but they may also denote a recommendation, for instance, or description of state. Each sentence has one main verb. An example of a verb SID is "Execute".

Verb SIDs, unlike most other SIDs, do not take a concrete data type for an argument. Instead they take a sequence of one or more S-expressions. These S-expressions describe the various "players" for the verb. In the case of "Execute", we would be interested in what (program) was executed, who executed it, where and when it was executed, and so on.

2.4.3. Role SIDs

A verb has little value until we describe who and what that verb applies to. This is accomplished using *role* SIDs. A role denotes what part an entity, or set of attributes, plays in a sentence. Examples of roles are "Initiator" and "Operand".

Role SIDs, like verb SIDs, take a sequence of one or more S-expressions as argument. These S-expressions describe the object, roughly speaking, which is playing that role in the sentence.

Example:

```
(Initiator (RealName "Joe Cool") (UserName "joe") (UserID "1618"))
```

denotes a user, with real name "Joe Cool", user name "joe", and user ID "1618", acting as Initiator. (Typically, an Initiator is someone who causes an action to take place--such as executing a program.)

An S-expression headed by a role SID is called a *role clause*.

2.4.4. Extension SIDs

It is not expected that any component will understand all SIDs. A component concerned with Unix notions will often not be worried about X.500-related SIDs. Nevertheless, many X.500-related SIDs have their complements in the Unix world, and the Unix component will want to capture this information, even if it isn't cognizant of the exact use of this information in the X.500 world. For instance, a user's real name is a user's real name, although in Unix it might be the name in /etc/passwd associated with the user's account, and in X.500 it may be a Common Name. If these two concepts were expressed with two completely distinct SIDs, then we would lose much of the benefits of data sharing.

Extension SIDs are designed to address this. Extension SIDs allow one to specify information in a relatively generic fashion, and then give more specialized receivers extra information about a SID that specifies

more precisely how it is to be used. For instance, an X.500 Common Name would be expressed as follows:

```
(RealName (ExtendedBy X500CommonName) "Joe Cool")
```

Most components would be able to understand the RealName SID, and would be able to capture the fact that the a user with the real name "Joe Cool" is in question here. Additionally, any component who understands X.500 **would implement the X500CommonName extension, so that it knows** that the real name is registered as a Common Name, along with any implications of that fact.

In general, a SID is *extended* by following it with a sequence of one or more SID-pairs, each of which is tagged with the ExtendedBy SID. An extension SID MUST follow the SID or extension which it extends. For example, the following is well-formed:

```
(ObjectName (ExtendedBy DeviceName) (Extendedby UnixFullDeviceName)
    ...
)
```

where the ellipsis indicates the sequence of S-expressions qualifying the ChangePrivilege verb.

An extended SID always takes the same *type* as the unextended (base) SID. In fact, if one knows that a message will *only* be used by someone who recognizes the extension, then it may omit the base class altogether, and refer only to the extension. Therefore, for instance, one could write

```
(X500CommonName "Joe Cool")
```

2.4.5. Conjunction SIDs

Conjunction SIDs join sentences at the same "level" together. Two sentences that are simply juxtaposed together are presumed to mean that both hold. That is,

```
<Sentence1> <Sentence2>
```

means that both Sentence1 and Sentence2 hold. Other relationships are indicated by the appropriate conjunction SID. For instance, to indicate that Sentence1, Sentence2, and Sentence3 all had a common cause, one writes

```
(CommonCause <Sentence1> <Sentence2> <Sentence3>)
```

2.4.6. Open S-Expressions

An open S-expression is one in which not all the data values are "filled in", so to speak. It is used to express concepts such as "<Someone> removed <some file>." Its only currently defined usage is in the def construct, as follows:

```
(def RemoveFile ($username $filename)
    (Remove
        (Initiator (UserName $username))
        (Operand (ObjectType file) (ObjectName $filename))
    )
)
```

In later usage, we can express "The user with user name joe removed the file /etc/passwd" in this way:

```
(RemoveFile "joe" "/etc/passwd")
```

Its general format is

```
(def <NewSID> (<ListOfArguments>) <SIDExpansion>)
```

2.4.7. Referent SIDs

There is a last special type of SIDs, called Referent SIDs. They are placed at the end of this chapter, because they are not restricted to the construction of a single sentence, but instead allow one to link two or more sentences together (though they are often used to refer to other parts of the same sentence).

The two referent SIDs are ReferAs and ReferTo. They take a string as their data type. A SID-pair headed by a referent SID is called a *referent clause*. A referent clause may be placed into either a sentence or a role clause. Their interpretation varies depending on where they appear:

* If a ReferAs clause is placed into a sentence, it can be said to *refer* to that sentence, *except* for any ReferAs clauses. (It is considered bad form to use more than one ReferAs clause in the same sentence at the top level.) Thereafter, a use of the corresponding ReferTo clause can be used in place of that sentence (although see warning below).

* If a ReferAs clause is placed into a role clause, it is said to refer to the object described by the sequence of S-expressions following that role, *except* for any ReferAs clauses. (It is considered bad form to use more than one ReferAs clause in the same role clause.) Thereafter, a use of the corresponding ReferTo clause can be used in place of that object description (again, see warning below).

* WARNING. The referent SIDs MAY carry actual semantics, and are not simply macros. If a ReferAs clause is placed into a sentence, and that sentence refers to an event (say), then the ReferTo clause refers specifically to that specific event, and not simply to an event with the same attributes (which after all may not be uniquely identifying). Similarly, if a ReferAs clause is placed into a role clause, and that role clause describes an object (say) then the ReferTo clause refers specifically to the same object, and not simply to an object with the same attributes.

  Of course, if no specific item is denoted by the ReferAs clause, then this warning does not apply. For example, if ReferAs occurs in an assertion of state, then it can be interpreted as simply a macro, since there is no unique item being denoted.

As an example, consider the following sequence:

```
(Remove
    (Initiator (RealName "Joe Cool"))
    (Operand (FileName (ExtendedBy UnixPathName) "/etc/passwd"))
    (AtTime (Time "1998 Feb 25 12:40:32 PST"))
    (ReferAs "JoesDeletion")
)
```

followed by

```
(HelpedCause
    (ReferTo "JoesDeletion")
    (Login
        (Initiator (RealName "Mary Worth"))
        (To (HostName "host.work.com"))
        (Outcome (ExtendedBy UnixErrno) (ReturnCode 13))
    )
)
```


This indicates that the act of Joe Cool deleting /etc/passwd later helped to prevent Mary Worth from logging in to host.work.com. Note that this specific instance of Joe Cool deleting /etc/passwd is referred to here. Even if (by resetting the clock, say) Joe Cool were to delete /etc/passwd a second time with the same attributes, this construction would still show that it was the *first* deletion that helped prevent Mary Worth from logging in.

Since referent SIDs act across GIDOs, and hence potentially across multiple messages (although not necessarily so), the question of scope arises. The scope rule applying to Referent SIDs is as follows: The value of a referent clause is the verb or role within which it is found (roughly speaking), provided that that verb or role is in the same

thread. A thread is defined as the conjunction of the originator ID and thread ID fields in the GIDO header. A producer MUST NOT re-use a referent (such as "JoesDeletion") within the same thread, for perpetuity.

2.4.8. Guidelines for Putting SIDs Together to Form Sentences

In this section, we describe how to use verb SIDs, role SIDs, conjunction SIDs, and other kinds of SIDs to construct sentences.

2.4.8.1. Basic Organization

As noted above, a simple sentence is an S-expression headed by a verb SID (which may be extended). This verb SID is followed by a sequence of one or more S-expressions that describe the various entities that play parts in the sentence, or qualify the verb.

The S-expressions denoting the roles of the sentence are headed by a role SID, which may also be extended. This role SID is again followed by a sequence of one or more S-expressions that may describe attributes of the entity playing that role. It may also describe a sentence that plays a role within the sentence.

A BNF-like grammar that specifies this structure is as follows.


```
<SentenceList>      ::= <Sentence> <SentenceList>
                      | <Sentence>
<Sentence>          ::= "(" <ConjunctionSID> <ExtensionList>
     <SentenceList> ")"
                      | "(" <VerbSID> <ExtensionList> <QualifierList> ")
                      | "(" <VerbSID> <ExtensionList> <ReferToClause> ")
                      | "(" "def" <NewSID> <ArgList> <SIDExpansion> ")"
<QualifierList>     ::= <Qualifier> <QualifierList>
                      | <Qualifier>
<Qualifier>         ::= "(" <RoleSID> <ExtensionList> <QualifierList> ")
                      | "(" <RoleSID> <ExtensionList> <ReferToClause> ")
                      | "(" <AtomSID> <ExtensionList> <AtomSIDData> ")"
                      | "(" "ReferAs" <Referent> ")"
<ExtensionList>     ::= <Extension> <ExtensionList>
                      | <NULL>
<Extension>         ::= "(" "ExtendedBy" <ExtensionSID> ")"
<ArgList>           ::= <Arg> <ArgList>
                      | <NULL>
<ReferToClause>     ::= "(" "ReferTo" <Referent> ")"
```

In English: A GIDO payload is a SentenceList, which is a list of Sentences.

A Sentence may be a ConjunctionSID, followed by a list of the Sentences

it conjoins, or it may be a VerbSID, followed by a list of Qualifiers of that VerbSID.

A Qualifier may be a RoleSID, followed by a list of Qualifiers of that RoleSID. A Qualifier may also be an AtomSID followed by its data.

Any list of Qualifiers may contain a ReferAs clause. Thereafter, use of the corresponding ReferTo clause may stand in for that list of Qualifiers.

Any SID may be followed by a list of Extensions.

2.4.8.2. Understanding Sentences and the Principle of Connectedness

The Principle of Connectedness simply states that when a component reading a GIDO encounters a SID it does not understand, the component must strictly ignore the S-expression that the SID heads. The component MUST NOT reject the GIDO on this ground. For instance, in the example below

```
(InOrder
    (Delete
        (Initiator (FullName "Joe Hacker"))
        (Operand (ObjectType file) (ObjectName "/etc/passwd"))
    )
    (Execute
        (Initiator (UserName "sysadmin"))
        (Operand (ObjectType program) (ProgramName "SystemCheck"))
    )
)
```


if a component does not understand the Delete verb SID, it may not make use of the Initiator and Operand SIDs within that sentence, even if it understands those, because it will not understand what they are the Initiator and Operand *of*.

This is called the Principle of Connectedness because the portion of the GIDO which is understood must form a connected tree. If a parent is not understood, its children should not be interpreted, as its relation to the portion of the tree contain the parent is unknown.

2.4.8.3. Rules and Guidelines for Using SIDs

Whenever a component puts a SID into a GIDO, the SID MUST be used with the number of arguments (usually one) that the SID's definition calls for (see the definitions in Appendix B). The SID's argument(s) MUST have the syntax and meaning that the SID's definition calls for. Otherwise the component is OUT OF CONFORMANCE with the SID's definition.

A component that generates GIDOs MUST generate them in conformance with all of the SID definitions in this specification.

Whenever the above rule permits, a component generating a GIDO SHOULD use a SID from this specification and SHOULD avoid the SIDs defined in the Uninterpreted SIDs section. If the only suitable SID in this specification is in the Uninterpreted SIDs section, then an implementation MAY use it or define a new SID; defining a new SID is usually better.

If a component generating GIDOs uses a SID from a particular specification, and if that specification defines two applicable SIDs, one of which is strictly more specific than another, then the component SHOULD use the more specific one.

If CIDF component X creates a sentence and CIDF component Y later has a copy of the sentence and passes it verbatim to CIDF component Z, then Y MAY do so even if the sentence violates the above rules and guidelines. The sentence MUST be passed verbatim and SHOULD be clearly ascribed to its originator. This provision frees D-boxes and such from having to thoroughly understand and validate every GIDO they process.

However, if the CIDF component modifies any part of the sentence, then it is responsible for the sentence's compliance with the above rules and guidelines.

2.5. Detailed Examples

Now that the basic components of an S-expression have been presented, we illustrate how to utilize these components to express various records structures and messages that intrusion detection systems may wish to express. In the following examples, we walk the reader through the process of translating raw event structures, analysis results, and other candidate message structures into S-expressions.

2.5.1. Translating a Basic Security Module Audit Record

One very well-known form of security audit records are those introduced in Sun Microsystems' SunOS 4.1.X Basic Security Module (BSM). There are a variety of ways to translate BSM audit records into S-expressions, depending on the data elements that a CIDF module may be directed to filter or incorporate within its GIDOs. In this example we demonstrate the translation of a BSM audit record generated as a result of a successful rlogin request.

2.5.1.1. BSM Record Description

The raw BSM record describes an event in which an external user performs a successful remote login to target.machine.com from source.machine.com. A session is established in which the resulting real and effective user IDs are set to thomas, the real and effective group IDs are set to staff, terminal 6 is assigned to the session, and the process and session IDs are set to 5345.

The event is captured by the audit daemon on target.machine.com, which records the event as follows:

Raw BSM Audit Record

```
[header,86,2,login - rlogin,,Sat Jul 29 20:43:01 1995,
+ 280009000 msec subject,thomas,thomas,staff,thomas,staff,
5345,5345,0 6, source.machine.com text,successful login return,
success,0]
```

2.5.1.2. BSM to S-Expression Translation Process

Now we illustrate the underlying rationale used to translate a common event structure such as a BSM audit record into a CIDF S-Expression. As discussed in Section 3.2, we begin our S-expression construction by first defining the verb of our sentence in its most general form. In this case, the operation recorded in the BSM audit record is the establishment of a communication session between two entities via rlogin. As we parse the potential Verb SIDs available in Appendix B.2, we find that the SID most closely matching the rlogin operation is the BeginSession SID. While BeginSession captures well the underyling action represented in the audit record, we note that a Unix-specific extension is available for further refinement (as discussed in Section 3.4). The resulting S-expression is as follows:

Example 2.5.1.2a BSM Rlogin S-Expression:

```
- -->(BeginSession (ExtendedBy UnixRlogin)
                     :
                     :
                     :
- -->)
```

The next step is to qualify the verb with supporting S-expressions that further enumerate the attributes of the event. In this case, the verb BeginSession has a series of supporting role clauses that can be derived from the BSM record (Section 3.4). These role clauses include:

- the observer from which the event was recorded
- the initiator of the BeginSession operation
- the entity to whom the BeginSession was directed
- the resulting state changes or resource(s) produced or

destroyed by the operation (in our case this involves the attributes of the session established by the rlogin

- and the outcome of the event

>From the above categories of attributes we augment the S-expression with the following relevant role-clauses:

Example 2.5.1.2b BSM Rlogin S-Expression:

```
 (BeginSession (ExtendedBy UnixRlogin)
- -->      (Observer   (S-expression ...) )
- -->      (Initiator  (S-expression ...) )
- -->      (To         (S-expression ...) )
- -->      (Operand    (S-expression ...) )
- -->      (Outcome    (S-expression ...) )
  )
```

Role clauses are selected for grouping associated datafields under a common contextual usage in the S-expression sentence. At this point, we switch our attention to incorporating associated datafields within the above role clauses. Datafields that cannot correctly be associated within the context of one of the available role clauses can still be incorporated in the S-expression independent simple sentences within the S-expression.

In our example, the Observer clause provides a contextual association with all datafields that describe attributes of the oberserve, including when, where, and through which means (i.e., BSM data) the observation was recorded. The initiator clause is used to associate datafields that describe the entity responsible for the event. In this case, the BSM record provides very little information, other than hostname from which the request was sent. Similarly, the BSM record provides only the hostname of the recipient, which we document in the To clause. The Operand clause is used to describe object that has been affected by the event, which in this case was the creation of the session. - From the BSM audit record, we can include under the Operand clause the session's associated user attributes, group attributes, process/session attributes, and the device through which the session is supported. Lastl we enumerate the attributes of the outcome.

Example 2.5.1.2.c Final BSM Rlogin S-Expression:

```
   Section Ref.
   ------------
 (BeginSession (ExtendedBy UnixRlogin)                  -- B.2.5
     (Observer                                          -- B.3.7
- -->     (AtTime (Time "Sat Jul 29 20:43:01 PDT 1995"))   -- B.3.2
- -->     (HostName "target.machine.com")                  -- B.5.4
```

```
- -->     (ObservationSourceType "BSM-SunOS")              -- B.5.1
  )
 (Initiator                                               -- B.3.1
- -->   (HostName "source.machine.com")                    -- B.5.4
  )
(To                                                       -- B.3.3
- --> (HostName "target.machine.com")                      -- B.5.4
 )
 (Operand                                                 -- B.3.1
- -->   (UnixAUserName "thomas")                           -- B.5.9.6
- -->   (UnixUserName "thomas")                            --   "
- -->   (UnixEUserName "thomas")                           --   "
- -->   (UnixGroupName "staff")                            --   "
- -->   (UnixEGroupName "staff")                           --   "
- -->   (ProcessID 5345)                                   -- B.5.2
- -->   (SessionID 5345)                                   -- B.5.2
- -->   (Through                                          -- B.3.3
- -->        (ObjectName                                   -- B.5.1
- -->            (ExtendedBy UnixFullDeviceName)          -- B.5.1
- -->         "/dev/tty06")
      )
 )
 (Outcome                                                 -- B.3.6
- -->   (Severity 3)                                      -- B.5.1
- -->   (ReturnCode                                       -- B.5.1
- -->        (ExtendedBy UnixErrno)                       -- B.5.1
- -->    0)                                               -- B.5.1
- -->   (Comment "successful login")                       -- B.5.1
 )
)
```


2.5.2. Translating a TCP/IP Packet

In the next example, we'll see how to translate the contents of an FTP connection request captured by a TCP/IP packet sniffer. Here the TCP/IP packet is observed being sent from an external client to the target host's FTP control port. The packet is translated by a CIDF module that attempts to describe the transaction from the perspecti of analyzing data sent to the application-layer (i.e, FTP) network servi

2.5.2.1. TCP/IP Packet Description

The observer in this example is a CIDF E-box that parses sniffed pacekts from a Sun Microsystem's Solaris machine. The observer's host platform i named snoopmachine.machine.com, and from this machine the observer attem to capture and translate traffic to and from the FTP control port of server.machine.com using the Solaris snoop(1) command:

```
 snoopmachine% snoop -v -d le0 -t a host server port 21
```

The following is an example snoop-formatted packet produced be the observer:

Raw TCP/IP Packet


```
ETHER: ----- Ether Header -----
ETHER:
ETHER: Packet 7 arrived at 8:59:49.05
ETHER: Packet size = 70 bytes
ETHER: Destination = 0:01:02:03:04:05, Western Digital
ETHER: Source = 0:aa:bb:cc:dd:ee,
ETHER: Ethertype = 0800 (IP)
ETHER:
IP: ----- IP Header -----
IP:
IP: Version = 4
IP: Header length = 20 bytes
IP: Type of service = 0x00
IP:     xxx. .... = 0 (precedence)
IP:     ...0 .... = normal delay
IP:     .... 0... = normal throughput
IP:     .... .0.. = normal reliability
IP: Total length = 56 bytes
IP: Identification = 63187
IP: Flags = 0x4
IP:     .1.. .... = do not fragment
IP:     ..0. .... = last fragment
IP: Fragment offset = 0 bytes
IP: Time to live = 38 seconds/hops
IP: Protocol = 6 (TCP)
IP: Header checksum = 69a3
IP: Source address = 999.998.997.996, client.machine.com
IP: Destination address = 111.121.131.141, server.machine.com
IP: No options
IP:
TCP: ----- TCP Header -----
TCP:
TCP: Source port = 12406
TCP: Destination port = 21 (FTP)
TCP: Sequence number = 820300070
TCP: Acknowledgement number = 3095138926
TCP: Data offset = 20 bytes
TCP: Flags = 0x18
TCP: ..0. .... = No urgent pointer
TCP: ...1 .... = Acknowledgement
TCP: .... 1... = Push
```

```
TCP: .... .0.. = No reset
TCP: .... ..0. = No Syn
TCP: .... ...0 = No Fin
TCP: Window = 61320
TCP: Checksum = 0x4e8d
TCP: Urgent pointer = 0
TCP: No options
TCP:
FTP: ----- FTP: -----
FTP: "USER anonymous\r\n"
```


The packet consists for four layers of structure: the Ethernet header, the IP header, the TCP header, and the FTP data portion. Working from the bottom up, we see that the packet represents an FTP "USER anonymous" request, which for FTP is equivalent to a BeginSession request for an anonymous FTP session. Above the FTP header are the TCP fields, containing, among other things, the source and destination ports (note the destination port is port 21, the FTP control protocol port). Above the TCP layer are the IP and Ethernet header, both containing datafields that could be of use to further identify the initiator and recipient of the FTP request.

2.5.2.2. TCP/IP Packet to S-Expression Translation Process

As with the BSM exaample, we begin our S-expression by defining the verb of our sentence. In this example, the E-box is monitoring traffic to the FTP control port when it encouters a TCP/IP packet that contains an FTP USER command request for anonymous access. As a result, we again choose BeginSession as the verb. The resulting S-expression is as follows:

Example 2.5.2.2a FTP BeginSession S-Expression Example:

```
- --> (BeginSession
                     :
                     :
                     :
- --> )
```

Next, we qualify the verb with supporting S-expressions that further enumerate the attributes of the event. As with BeginSession in our BSM example, we can support a series of role clauses from the information in our FTP packet. These role clauses include:

- o the observer from which the event was recorded
- o the initiator of the BeginSession operation
- o the entity to whom the BeginSession was directed
- o the resulting state changes or resource(s) produced or

destroyed by the operation (in our case this involves the attributes of the session established by the rlogin
 o the command or tool used in the event
 o and the outcome of the event

- From the above categories of attributes, we augment the S-expression with the following relevant role-clauses:

Example 2.5.2.2b FTP BeginSession S-Expression Example:

```
 (BeginSession
- -->             (Observer  (S-expression ...) )
- -->             (Initiator (S-expression ...) )
- -->             (To        (S-expression ...) )
- -->             (Operand   (S-expression ...) )
- -->             (Using     (S-expression ...) )
- -->             (Outcome   (S-expression ...) )
 )
```


The Observer clause can include a variety of datafield attributes, including the timestamp and the host platform of the sniffer. The initiator of the BeginSession could also be viewed as attributes of the location from which the request was sent. Because both the "Initiator" and "From" roles both provide accurate context to the set of attributes that represent the entity responsible for the BeginSession Event, we chose to recognize the two clause using referent SIDS (Section 3.7). The entity responsible for the event can be described through a variety of attributes within the packet, including the Ethernet address, IP address, TCPPort, and hostname. The recipient can be identified from a similar set of corresponding datafields. Unlike the BSM record, there is very little information in the packet to describe the session, other than the session will be associated with the anonymous user account. The means used in this event is an FTP command, "USER".

Lastly, we identify the outcome of this event as pending, in that at this point we cannot determine whether the BeginSession will succeed. The outcome will be determined in subsequent GIDOs, which require an association with this S-expression through a common thread ID define in their GIDO headers. We use the CIDFReturnCode extension of ReturnCode to express this condition. The GIDO recipient must consult the other GIDOs in the thread until it encounters an Outcome with a ReturnCode that is not pending.

Example 2.5.2.2.c Final FTP BeginSession S-Expression:

Section Ref.

```
                                                              ------------
 (BeginSession (ExtendedBy FtpCommand) "USER"                 -- B.2.5
     (Observer                                                -- B.3.7
- -->     (AtTime (Time "08:59:49.1 PDT"))                      -- B.3.2
- -->     (HostName "snoopmachine.machine.com")                 -- B.5.4
- -->     (ObservationSourceType "Packet")                      -- B.5.1
     )
     (Initiator                                               -- B.3.1
- -->     (ReferTo "the-client")                                -- B.5.1
     )
     (From
- -->     (ReferAs "the-client")                                -- B.5.1
- -->     (HostName client.machine.com)                         -- B.5.5
- -->     (EthernetAddress 0:aa:bb:cc:dd:ee)                    -- B.5.5.1
- -->     (IPv4Address 999.998.997.999)                         -- B.5.5.2
- -->     (TCPPort 12406)                                       -- B.5.5.3
     )
     (To                                                      -- B.3.3
- -->     (EthernetAddress 0:01:02:03:04:05)                    -- B.5.5.1
- -->     (IPv4Address 111.121.131.141)                         -- B.5.5.2
- -->     (Hostname "server.machine.com")                       -- B.5.5
- -->     (TCPPort 21)                                          -- B.5.5.3
     )
     (Operand                                                 -- B.3.1
- -->     (UserName                                             -- B.5.4
- -->        (ExtendedBy UnixUserName)                          -- B.5.9.6
- -->     "anonymous")
     )
     (Using                                                   -- B.3.1
- -->     (FTPCommand "USER")                                   -- B.5.9.5
     )
     (Outcome                                                 -- B.3.6
- -->     (ReturnCode                                           -- B.5.1
- -->        (ExtendedBy CIDFReturnCode)                        -- B.5.1
- -->     pending)
     )
 )
```

2.6 Rules and Guidelines for Defining SIDs

Other specifications MAY define SIDs for use with the CIDF framework. If a CIDF component generates or uses those SIDs, those SIDs MUST be defined in conformance to the rules here and SHOULD be defined in conformance with the guidelines here.

- Every SID MUST have a unique name.
- Every SID's definition MUST include precise syntax.
- Every SID's definition SHOULD include precise semantics.
- The SID description must fully explain the intended use of SID (i.e., the intended data arguments must be described)

```
# Editor's note: The Event Subgroup is investigating naming
# conventions and rules for SID enumeration to eliminate the
# potential for SID reuse.
```

Specifiers SHOULD avoid defining a SID whose meaning overlaps another, unless one SID is strictly more specific than another (unless the first one provides all the information that the second one provides and more).

A SID MUST be so defined that when the SID heads an S-expression, the truth of its S-expression is independent of the peer S-expressions, the containing S-expression's peers, the peers of the container of the containing S-expression, and so on.

Thus, an S-expression cannot *modify* the meaning of a peer S-expression. It can only augment the the peer S-expression. (The logical relationship between peer S-expressions is conjunction.) This is critical because a consumer may ignore some peer S-expressions.

Specifiers should be wary when defining a set of closely related SIDs, since a consumer may understand some of the SIDs and not others. If two data items can be properly understood together but cannot be properly understood singly, then it is advisable to define a single SID that takes both data items as arguments.


```
================================================================================
================================================================================
=
=                              3: Encoding Gidos
=
================================================================================
================================================================================
=
================================================================================
= 3.1: Introduction to Gido Encoding
================================================================================
```

In encoding a gido into actual bytes for storage, tranmission, etc, two things are involved. Firstly, every gido is accompanied (in perpetuity) by a static format header which contains basic information about that gido. This header format is described in section 3.2.

Secondly, the S-expression which forms the payload of the gido must also be encoded. The method for doing this is covered in section 3.3

==========
= 3.2: Gido Header
==========

3.2.1: Introduction

The header definition, presented in this section, consists of a series of constant fields that gido consumers can reliably parse to read basic data common across all gidos. The gido s-expression payload, presented in a preceding section, contains the actual IDS component-specific data structures, including semantic identifiers that allow gido consumers to decode and interpret individual fields.

The gido header is used to convey information about the gido itself, rather than details of the event, analysis report, or response prescription (which are captured in the payload). Each CIDF-compliant gido generated by any component MUST contain these fields in this order (for this version). Consult Appendix A for details on type definition.

3.2.2: The Header Fields

1. **Version ID (type revision).** Indicates the format revision used to encode this gido. Initially, the Version ID will indicate CIDF Version 1.0 (major = 1, minor = 0). This Version ID will be incremented as future versions are introduced. All current and future versions of this specification must reserve the first field of the gido header for the Version ID. Gido consumers may reliably use this field to detect the format of the remainder of the gido.

##########
Editor's Comment: This field suggests that CIDF revision
identifiers will follow a major.minor format. The CIDF working
group must decide if this is the proper revision format, and must
then define the meaning of major and minor revision indicators.
##########

2. **Gido Length (4 octets, big-endian).** Indicates the byte length of the entire gido, including this header but excluding any optional digital signature. This field may be used to cross-check gido completeness.

3. **Time Stamp (4 octets, big-endian).** Indicates the seconds since Unix epoch 1970. This time refers to the moment that this report or request was generated. Specifically, it does not refer to the time that any events were first detected, or when they occurred; these (if they are known) are to be placed in the message payload itself.

4. **Thread ID (4 octets).** Used to identify gidos with some

common thread; all gidos about a given event (e.g., first report followed by successive updates) would share the same Thread ID.

5. **Class ID (2 octets).** Indicates the category that the event, analysis, or response generator believes the gido falls under. Class IDs are defined in Section 3.2.3. This field is intended to allow receivers to process high-priority gidos in a given field of expertise before all others. Note that some codes are reserved for user-defined Class IDs; the receiver must check to see if prior agreement exists between sender and receiver on these codes.

6. **Originator ID (unknown type).** A unique identifier associated with the component generating this gido.

```
########################################################################
# Editor's Comment:  The format and semantics of the Originator ID
# is an open issue that requires resolution by the CIDF working group.
# Specifically, how will CIDF modules be uniquely identified from other
# CIDF modules?
########################################################################
```

7. **Flags (1 octet).** The bits of this flags octet are to be interpreted according to the following table:

Bit	Meaning
0 (LSB)	set = optional signature present (see below). clear = no optional signature
1-7 (MSB)	reserved (MSB = most significant bit)

The gido payload, plain or compressed, immediately follows the header. If bit 0 in Flags is cleared, indicating no optional signature, the gido ends with the payload (indicated by the Gido Length header field). Otherwise, if bit 0 is set, indicating that a digital signature of the content is present, this signature is contained in a structure following the gido payload. Recall that the Gido Length header field indicates the end of the gido payload, not including the signature structure.

The signature structure has the following fields in it:

1. **Signature Length (2 octets).** Indicates the length, including this field (signature length), of the signature structure, in octets.

2. **Key ID (type unknown).** Uniquely identifies the key used to generate the signature. This ID may be understood only by a given receiver if the gido is to be sent one-to-one. This field also implies the signature algorithm.

```
########################################################################
# Editor's Comment: This issues is tied up with that of originator-id
#
########################################################################
```


3. Signature data. The entire gido represented by the Gido Length header field is passed through a gido digest, resulting in a short, fixed-length quantity. This quantity is then signed using the applicable encryption/signature algorithm, and the result of this operation placed in this field.

3.2.3 Class ID Codes

The following default Class ID codes are defined for events and analysis results. Under this scheme, class ids 0 thru 15 are reserved for CIDF event priorities, and 16 thru 31 are reserved for analysis report priorities. In addition, class ids 32 thru 127 are reserved for future CIDF extensions. IDS developers may use the remaining range (128 thru 255) for application-specific purposes.

```
                        (Default Event Class IDs)
        00 - Complete Event
        01 - Intermediate Event
        02 - Incomplete Event
        03 - E-box Internal Error Report
        04 - E-box Internal Warning Report
        05 - E-box Internal Status Message
        06 - Reserved for E
        07 - Reserved for E
           :
        15 - Reserved for
                    (Default Analysis Class IDs)
        16 - Critical Security Violation
        17 - Potential Security Violation
        18 - Suspicious Report
        19 - Warning Report
        20 - Intermediate Result
        21 - Informational Report
        22 - A-box Internal Error Report
        23 - A-box Internal Warning Report
        24 - Reserved for A
        25 - Reserved for A
           :
        31 - Reserved for A
                    (Reserved Priority Code Range)
########################################################################
# Editor's Comment:  Class ID code range 32-48 is reserved for
```

```
# R-Box countermeasure directives.
#########################################################################
        32 - Reserved for future use
        33 - Reserved for future use
           :
        127 - Reserved for future use
                        (Undefined Priority Codes)
        128 - Undefined
           :
           : (Undefined values may be employed for
           :  application-specific purposes.)

255 - Undefined
```


3.3: Encoding S-Expressions

GIDO payloads consist of S-expressions. However, these S-expressions are translated to an octet encoding format for efficient transmission or storage.

The octet encoding of message payloads support highly efficient transmissions of messages. This section describes how to transform an S-expression into the appropriate octet encoding. This encoding is designed to meet the following objectives:

* It must indicate the structure, so that a component ignorant of the elements within the S-expressions will still be able to parse the S-expressions.
* It must allow for pre-defined and distributed-out-of-band SIDs.
* It should be as compact as possible.

3.3.1: Octet Codes

The following codes will be used to represent various octet values in the succeeding encoding specifications. They are *not* S-expression atoms.

Code	Value	Interpretation
SEP	0xff	Used as separator.
SOPEN	0xfe	S-expression open.
PTR	0xfd	Pointer (referred to as @).
SID	0xfc	Prelude to SID 2-octet code.
TYPE	0xfb	Indicates concrete syntax type.

3.3.2: Encoding of S-Expression Grammar

What follows is the grammar for CIDF S-expressions. After each line we give the encoding applicable to that line.

```
    <item-list> ::= <item>
        E(<item-list>) = E(<item>)

    <item-list> ::= ( <item-list> )
        E(<item-list>) = E(<item-list>)

    <item-list> ::= <item-list> <item>
        E(<item-list>) = E(<item-list>) E(<item>)

    <item> ::= ( <sid-exp> <data-exp-list> )
        E(<item>) = SOPEN E(length{E(<sid-exp>) E(<data-exp-list>)})
                                  E(<sid-exp>) E(<data-exp-list>)
        E(length{X}) = var_encode(X)
```



```
    <item> ::= ( @ <sid-exp> <data-locator> )
        E(<item>) = SOPEN PTR E(<sid-exp>) E(<data-locator>)
        E(<data-locator>) = ascii_encode(<data-locator>)

    <item> ::= ( def <sid> <sid-exp> <semantics> )
        E(<item>) = SOPEN
                    E(length{E(def) E(<sid>) E(<sid-exp>) E(<semantics>)
                             E(def) E(<sid>) E(<sid-exp>) E(<semantics>)
        E(<sid>) = SID sid_encode(<sid>)
        E(<semantics>) = ascii_encode(<semantics>)

    <sid-exp> ::= <sid>
        E(<sid-exp>) = sid_encode(<sid>)

    <sid-exp> ::= '<type>:<sid>
        E(<sid-exp>) = TYPE type_encode(<type>) sid_encode(<sid>)

    <sid-exp> ::= ( <sid-exp-list> )
        E(<sid-exp>) = SOPEN E(length{E(<sid-exp-list>)})
                                    E(<sid-exp-list>)

    <sid-exp-list> ::= <sid-exp>
        E(<sid-exp-list>) = E(<sid-exp>)

    <sid-exp-list> ::= <sid-exp-list> <sid-exp>
        E(<sid-exp-list>) = E(<sid-exp-list>) E(<sid-exp>)

    <data-exp-list> ::= <data-exp>
        E(<data-exp-list>) = E(<data-exp>)

    <data-exp-list> ::= <data-exp-list> <data-exp>
```

```
        E(<data-exp-list>) = E(<data-exp-list>) E(<data-exp>)

    <data-exp> ::= <data>
        E(<data-exp>) = E(<data>)

    <data-exp> ::= ( <sid-exp> <data-exp-list> )
        E(<data-exp>) = SOPEN E(length{E(<sid-exp>) E(<data-exp-list>)})
                                    E(<sid-exp>) E(<data-exp-list>)
```

3.3.3: Auxiliary Functions

The following functions are used in the above syntax and encoding:

```
    ascii_encode(<string>) returns the ASCII-encoding of <string>.
    short_encode(<short>) returns the big-endian expression of
        <short>.  (E.g., short_encode(1234) = 0xd204.)
    sid_encode(<sid>) returns the 2-octet code for <sid>.
    type_encode(<type>) returns the SEP-terminated code for <type>.
```

var_encode(<int>) encodes an arbitrarily long integer. It is encoded as follows:

```
    L1 | <int>
```


where L1 is one byte containing the length of <int>, which is expressed in big-endian order.

3.3.4: Encoding Data

Data may be encoded in one of two ways. If the applicable SID had a fixed-length data type, then the data is encoded exactly as specified by the type; e.g., a ulong is encoded as four octets in big-endian order.

Otherwise, the data is encoded as follows:

```
    var_encode(length{Data}) | Data
```

Thus, if Data is a variable-length data structure that is 84,000 bytes long, then it is encoded as follows:

```
    03 01 48 20 xx xx xx ...
```

3.3.5: SID Codes

SIDs are ordinarily encoded as 2-octet values. A list of pre-defined SIDs is given in Appendix B; if one exists for the purpose, it SHOULD be used. However, this encoding furnishes the ability to define new SIDs should no applicable one exist, using the "def" operative. For the purposes of encoding, "def" is treated as a SID as well (i.e., it has

its own 2-octet code).

As noted in Section 3.3.2, this requires one to define a new SID code. These SID codes may be unrestricted, but they should conform to the following standard:

* The code is a 2-octet value, as stated above.
* The MSB (bit 7) of the first octet is the DYNAMIC bit. If this bit is set, this is a dynamically-defined SID, and the code for the actual SID is given by bit 5 of the first octet through the LSB (bit 0) of the second octet. If it is clear, this is a statically-defined SID, and the code for the SID is as given in the appendix.
* If the DYNAMIC bit is set, the 2-octet value is followed by a 4-octet value representing the UUID of the SID designer. Also, the next bit (bit 6 of the first octet) is the EXPERIMENTAL bit. If *this* bit is set, then the SID is ephemeral and cannot be relied on in future encodings. If it is clear, then this is a stable SID.

4: CIDF Communication

4.1: Message Layer

4.1.1: Rationale for Message Layer

The CIDF message layer was developed to solve problems of synchronization (i.e., blocking vs. non-blocking processes) and problems of different data formats for different operating systems. It also solves the problem that different groups will use different programming languages. In other words, the use of a messaging format achieves the following goals:

* Independent of blocking/non-blocking processes
* Data format independent
* Operating system independent
* Programming language independent

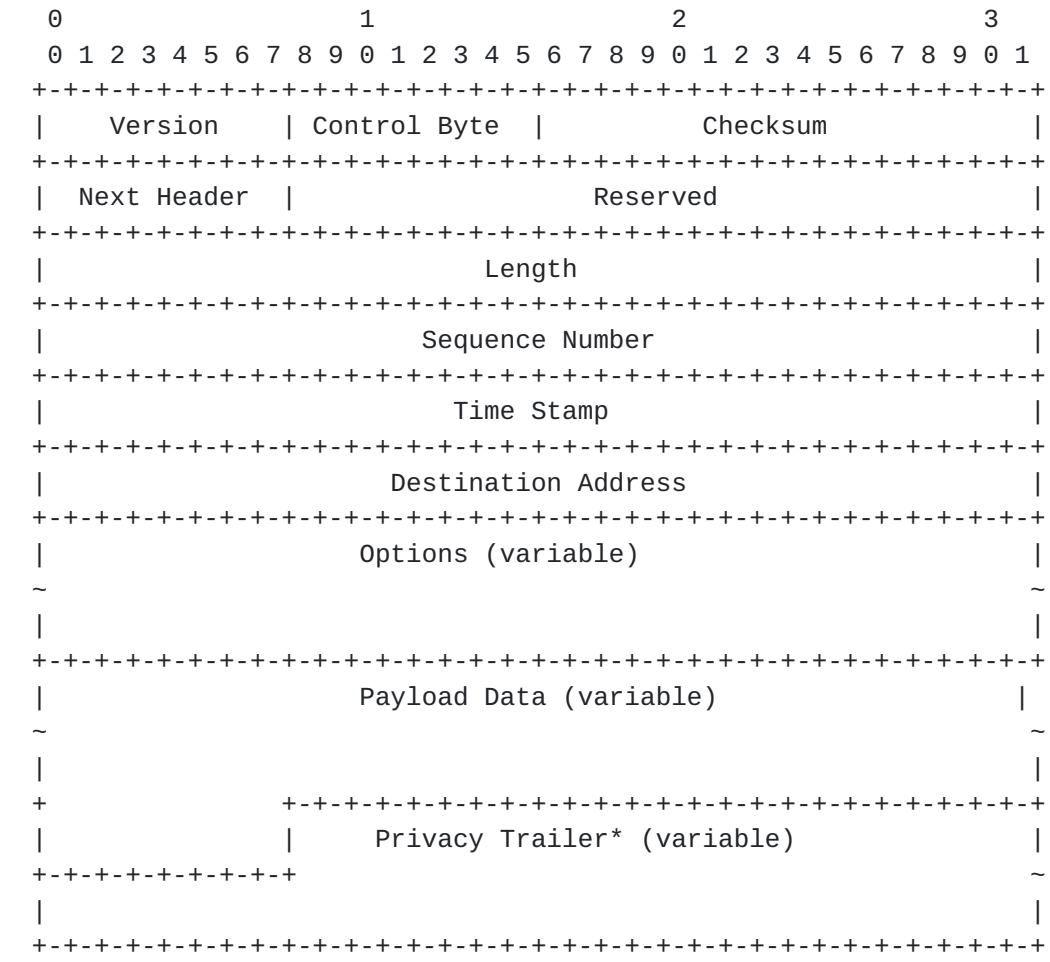
4.1.2: Objectives of the CIDF Message Layer

The top-level objectives for the CIDF message layer are to

* Provide an open architecture.
* Avoid imposing architectural constraints or assumptions on the systems or modules.
* Allow messaging independent of language, operating system, and network protocol.
* Support easy addition of new components to the CIDF.
* Support security requirements for authentication and privacy.
* Support devices that don't want to fully support CIDF.

4.1.3: Message Format

This message structure resides on top of the negotiated transport layer service. Note that all reserved fields are set to 0 on transmission and ignored on receipt.



* if privacy option is used

Options all have a common type-length-value format described below.

* Version - 1 octet. CIDF message-layer version (1 for this initial version).

* Control Byte - 1 octet. Used by the message layer to support reliable transmission, flow control, and security association management.
  - Acknowledgement of a delivered message (1).
  - Message received, but not delivered because of lack of resources (2).
  - Message received, but the supplied security association was not available to all processing (4).

* Checksum - 2 octets. A checksum across the entire CIDF message, prior to application of cryptographic mechanisms (i.e., privacy and authentication transforms). The checksum is computed as specified in the TCP standard (RFC 793).

* Next Header - 1 octet. Defines the type of either the next message layer option or application. The following are the currently defined types.
  - Application Header (1)
  - Route List (4)
  - Privacy Header (50)
  - Authentication Header (51)

* Length - 4 octets. Length of the CIDF message, including message header.

* Sequence Number - 4 octets. Message layer sequence number used for message reliability (acknowledgement and duplicate removal) and to support protection against message replay.

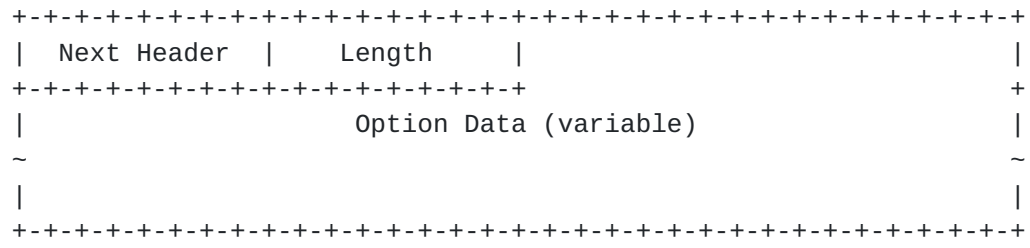
* Time Stamp - 4 octets. Used to provide loose time synchronization between CIDF communicating parties and to support tardy delivery detection (from denial of service).

* Destination Address - 4 octets. IP address of the target of this message. This field identifies the eventual recipient of the CIDF message and is used to route CIDF messages through intermediate CIDF nodes that cannot be traversed by normal network routing (e.g., firewalls).

4.1.4: Message Layer Protocol Options

Except for the CIDF privacy option, CIDF message options use the following format.

```
    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
```



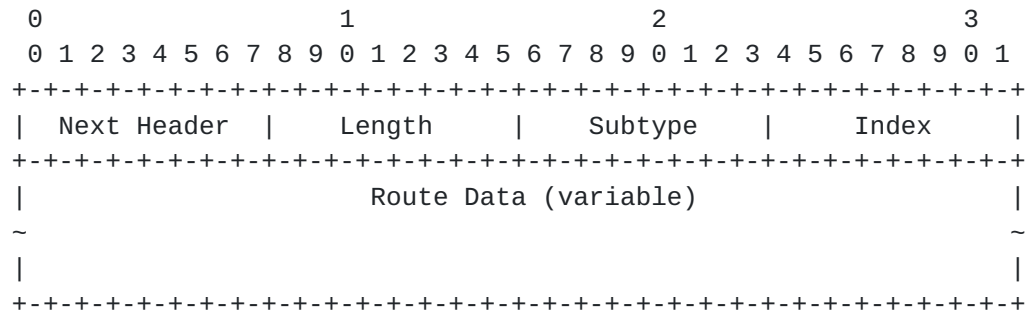
* Next Header - 1 octet. Defines the type of either the next message layer option or application, with the same permitted values as defined above.

* Length - 1 octet. Specifies the number of 32-bit words for this option, including the next type and length fields.

* Option Data - variable length. The option data field is always padded to a 32-bit aligned size.

4.1.4.1: Route List Option

Route List is a variable length field that specifies the CIDF nodes through which the message is to be routed for source routing, and through which the message has been routed for recorded routing. The Subtype field indicates whether this is a source or record route. The Route List has the following format. The route list option is used when the message destination and source are separated by CIDF nodes that cannot be traversed by normal network routing (e.g., firewalls).



* Next Header and Length are defined above.

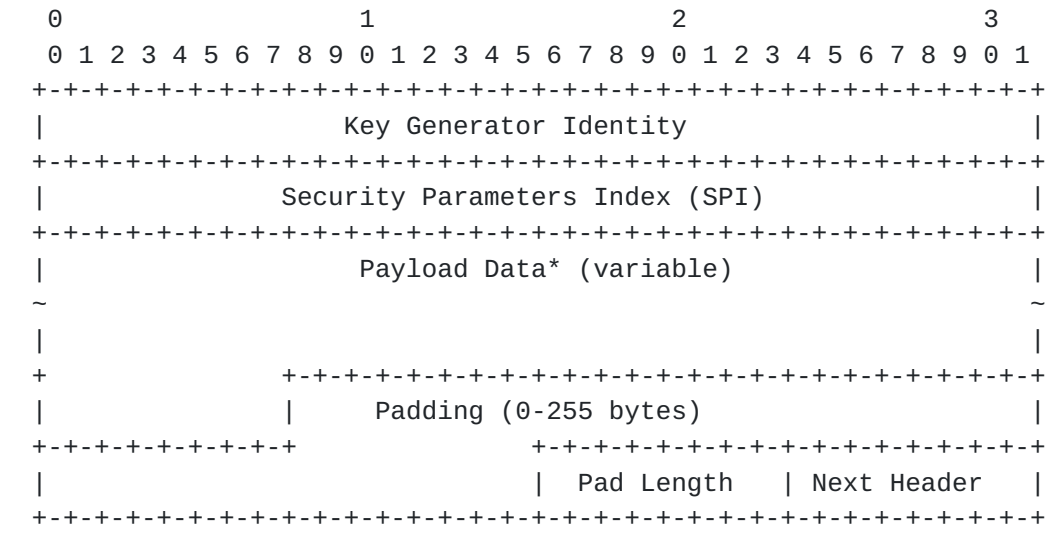
* Subtype - 1 octet. Specifies whether this is a recorded route or a source route.
  - Recorded Route (1)
  - Source Route (2)

* Index - 1 octet. Index into the array of addresses specifying the current address to be processed. For source routing, this is the address of the next CIDF hop. For recorded routes, this is the address of the last transmitting CIDF node.

* Route Data - variable length. This field is an array of Internet addresses. Each internet address is 32 bits or 4 octets. For a source route, if the index is greater than the length, the source route is empty and the routing is to be based on the destination address field. For a recorded route, if the index is greater than the length, the recorded route list is full.

4.1.4.2: Privacy Option

The CIDF privacy option supports both unicast or multicast privacy. For multicast privacy, one node of the multicast group is selected to generate the keys. The keys are then distributed to each multicast group member. For unicast privacy, each node generates its own privacy keys which are distributed to the remote party.



* (foot note) if the cryptographic algorithm requires use of an initialization vector, then that vector is placed as clear text between the SPI and Payload Data.

* Key Generator Identity - 4 octets. This value identifies the CIDF entity that generated the key. The initial use of this field is to specify either the key generator's IP address or for multicast applications the multicast address for the multicast group using this security association.

* Security Parameters Index (SPI) - 4 octets. The SPI is an arbitrary 32-bit value that uniquely identifies the Security Association for this message, relative to the key generator identity.

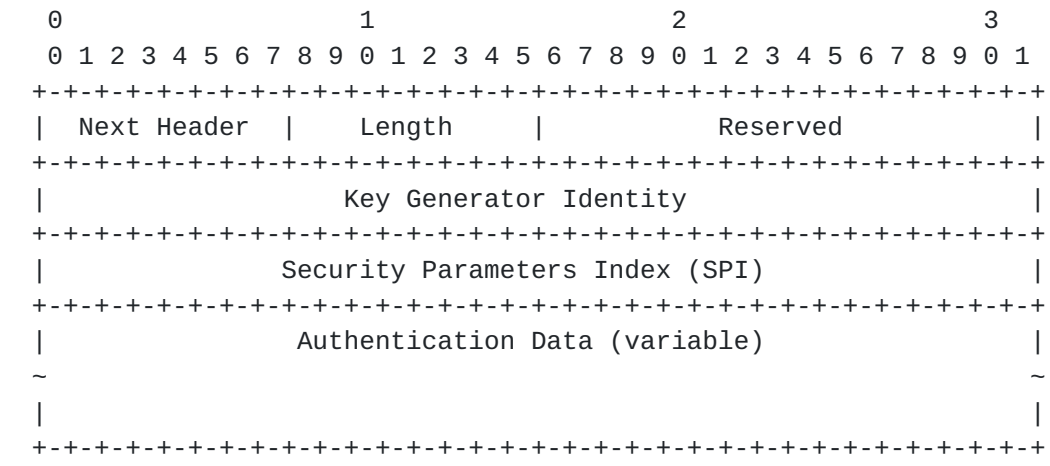
* Padding - variable length. The transmitter may add up to 255 bytes of padding if required to support the block size of the

cryptographic algorithm. Padding is required to ensure that after the privacy option is applied, the message ends on a 4-byte boundary.

* Pad Length - 1 octet. The number of padding bytes immediately preceding it. The range of valid values is 0-255, where a value of zero indicates that no Padding bytes are present.

* Next Header is defined above.

4.1.4.3: Authentication Header Option



* Next Header and Length are defined above.

* Key Generator Identity - 4 octets. This value identifies the CIDF entity that generated the key. The initial use of this field is to specify either the key generator's IP address or for multicast applications the multicast address for the multicast group using this security association.

* Security Parameters Index (SPI) - 4 octets. The SPI is an arbitrary 32-bit value that uniquely identifies the Security Association for this message, relative to the key generator identity.

* Authentication Data - variable number of 32-bit words. The data (e.g., digital signature or keyed hash) used to provide cryptographic authentication.

4.1.5: Cryptographic Mechanisms

The CIDF message layer protocol provides data integrity and source authentication services for the negotiation phase of CIDF communication. This enables components to reliably establish communications with minimal security overhead. During the negotiation phase, the client and

server determine the specific cryptographic services to be provided for further communication.

The message layer provides the cryptographic mechanisms as options, enabling use of lower-level services (e.g., IPSEC), CIDF-specific mechanisms, or no cryptographic services, depending on application requirements.

The mechanisms used are determined by the client based on the mechanisms supported by the server. The message layer mechanisms provide the fields necessary to (1) determine the cryptographic services applied (if any), (2) determine the cryptographic context, and (3) provide timeliness and replay protection.

4.1.6: Negotiation Mechanism

4.1.6.1: Introduction

Our approach is to use the simplest reliable transport mechanism available (i.e., reliable CIDF messaging over UDP) as the default CIDF transport protocol. This simple protocol can then be used to negotiate a more or less complex protocol for those components requiring additional transport-layer services. This allows simple devices to participate easily, while allowing complex devices to take full advantage of other transport-layer mechanisms. The message layer provides optional services to compensate for weaknesses in the transport layer. The combination of the CIDF message layer with transport-layer options provides a range of communication capabilities that can be used to support different application requirements. The following types of transport/messaging are initially envisioned:

* No assured delivery over a connection-less transport. That is, the CIDF message layer without acknowledgement and retransmission directly over UDP.

* Assured delivery over a connection-less transport. That is, the CIDF message layer with reliable delivery (acknowledgement, retransmission, and duplicate removal) over UDP.

* Assured delivery over a connection-oriented transport. That is, the CIDF message layer directly over TCP.

* Object-oriented transport. That is, the CIDF operations over CORBA.

To enable support for components that must use minimal communication infrastructure, the default transport mechanism is based on UDP. The following sections define the default transport layer protocol, CIDF security services, and the transport negotiation mechanisms.

4.1.6.1.1: Rationale for negotiated transport layer

The simplest approach would be to mandate the use of a single transport protocol. But there is no one protocol that can adapt to the varying requirements of all anticipated CIDF applications. Depending on whether an application is concerned with real-time traffic or simple accrual of a database of events, different transport mechanisms are appropriate.

Specifically, some CIDF applications require a very light-weight communication channel that does not have the resource usage required by current TCP implementations, while other applications require a flexible and robust communication channel such as TCP. Other requirements include application-specific support for multicast, which is not supported by TCP. Therefore, we have requirements for connectionless communication, reliable connectionless communication, and reliable connection-oriented communication. Additionally, we have varying requirements for security services. In some applications and environments, the infrastructure provides adequate security services. In other applications, we require CIDF-layer security services for authentication, privacy, or both.

Nevertheless, communications clearly cannot begin between two specific components until a channel is agreed upon. At the very least, this implies that if we don't agree on a single channel for all transport, we need to agree on a single channel for transport negotiation.

This channel needs to be widely supported and freely available. Components are allowed to share data on whatever channel they wish, but they must support channel negotiation on the common mechanism.

To support this range of requirements we provide a protocol based on the reliable UDP variant of CIDF that enables applications to agree upon the desired transport protocol, plus the desired CIDF message-layer security services. This exchange is only necessary if the participants have not previously agreed upon a transport mechanism through external mechanisms (e.g., local configuration settings or through the CIDF directory service).

4.1.6.2: Default Transport Layer

The default transport layer protocol for CIDF messages is reliable CIDF messaging over UDP. Other transport layer protocols may be used following a negotiation using the default of protocols and services required and supported by the CIDF client and server. Until we acquire a well-known CIDF port number, we will use 0x0CDF as the CIDF port. The CIDF message layer will listen on the CIDF well-known port for incoming CIDF messages.

4.1.6.3: Conformant transport options

* CIDF message layer without acknowledgement and retransmission directly over UDP.

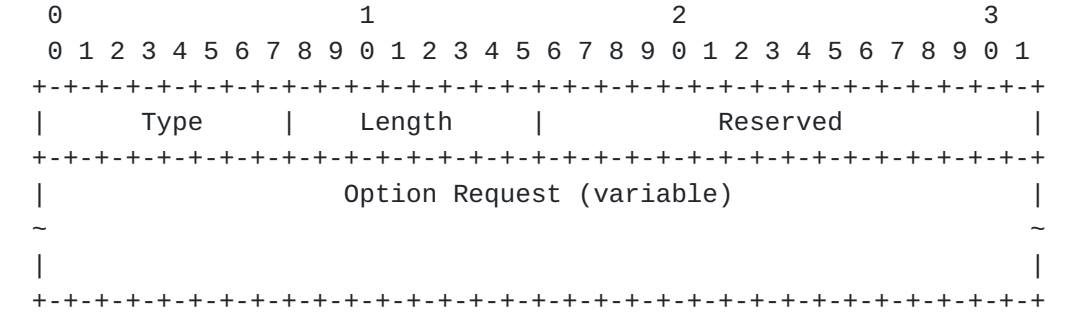
* CIDF message layer with acknowledgement and retransmission over UDP.

* CIDF message layer directly over TCP.

4.1.6.4: Option Negotiation Message Formats

The negotiation for more advanced communication services occurs over a UDP channel using only the CIDF message layer with authentication mechanisms enabled. This enables components that do not support TCP to participate in CIDF. Negotiation occurs by the client querying the server's capabilities. In response, the server specifies the class of CIDF operations supported, message services supported, and whether extensions are supported. The client then selects the services and message mechanisms. This information can also be provided by the directory server.

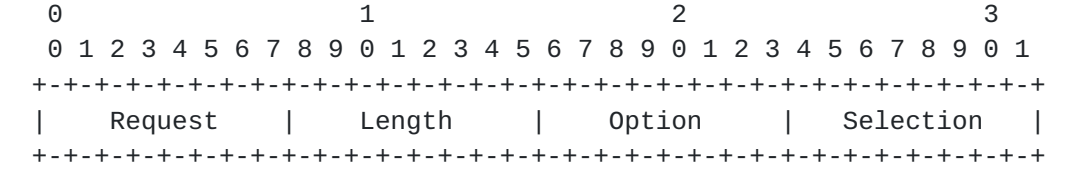
The CIDF transport negotiation protocol resides directly over the CIDF message layer. The query-response data format is shown below. We assume that for cryptographic services, the negotiation of the specific algorithms and modes is handled by the key distribution mechanism.



* Type - 1 octet. Specifies the type of request. For option negotiation messages, this value is 1.

* Length - 1 octet. Specifies the number of 32-bit words for this message, including the type and length fields.

Option Requests are formatted as follows.



```
|                  Option Parameters (variable)                 |
~                                                               ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

* Request - 1 octet. Specifies the type of request. The following request types are currently supported.
  - Want (1) - Preferred service.
  - Can (2) - Sender is capable of using this service.

* Length - 1 octet. Specifies the number of 32-bit words for this option request, including the request and length fields.

* Option - 1 octet. The option being negotiated. The following option types are currently supported.
  - Transport (1)
  - Privacy (2)
  - Authentication (3)

* Selection - 1 octet. The option value being negotiated. The meaning of this fields depends on the option being negotiated. The following selection values are currently supported.

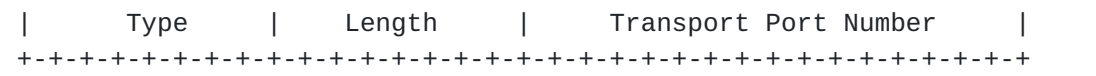
  For Transport negotiation.
  - None (0). Used to reject communication with another CIDF node when no acceptable options are received.
  - UDP (1)
  - Reliable UDP (2)
  - TCP (3)

  For Privacy negotiation.
  - None (0)
  - IPSEC (1)
  - SSL (2)
  - CIDF (3)

  For Authentication negotiation.
  - None (0)
  - IPSEC (1)
  - SSL (2)
  - CIDF (3)

Currently, the only option parameter specified is the selection of TCP/UDP port number for transport negotiation, which is formatted as follows.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```



* Type - 1 octet. Specifies the type of option parameter. For port numbers, this value is 1.

* Length - 1 octet. Specifies the number of 32-bit words for this message, including the type and length fields.

* Transport Port Number - 2 octets. This specifies on which port number the sender of the message will listen following completion of negotiation. Both ends of the channel select their own respective ports.

4.1.6.5: Protocol Description

Identification of the remote CIDF component's IP address is handled either through manual configuration or through the CIDF directory service. Note that this service may also indicate the CIDF component's capabilities (can) and preferences (want) for transport and security services.

When Sender S wishes to communicate with Receiver R, and the two components have not yet agreed on a transport mechanism, then S must initiate transport mechanism negotiation.

S sends a negotiation message to R on the CIDF well-known port indicating the services preferred (if any) and permitted. S includes a separate option request for each supported option, indicating the preferred option (if any).

When R receives an option negotiation, R selects the desired value using local preferences if supported by S, S's preferences if supported locally, or the intersection of local and S's capabilities if the preferences are not specified or supported.

If the local and remote capabilities do not permit communication, the R selects a transport option of None, indicating that communication is not feasible.

R responds with only the selected options for transport, privacy, and authentication identified as preferred options.

==========
= 4.2: Message Layer Processing
==========

4.2.1: Introduction

This section describes the processing of CIDF message layer messages. The standard procedures are used for CIDF messages independent of the transport layer. The reliable transmission procedures describe additional procedures to be used when the transport mechanisms is reliable UDP. CIDF privacy and authentication procedures describe the procedures used in providing CIDF layer privacy and authentication mechanisms, respectively.

4.2.2: Standard Procedures

Each CIDF message uses the standard CIDF header.

4.2.2.1: Outbound Message Processing

On request by the application layer to transmit a CIDF message, the CIDF message layer shall build the message header and append the message.

If the application indicates that this message requires source routing, then the CIDF message layer shall use the supplied source route list.

If the application indicates that this message requires recorded routing, then the CIDF message layer shall initialize the record route list, placing the outgoing IP address as the first entry on the route list.

The CIDF message layer shall insert the current CIDF version number, the application-provided destination, and the current time as the CIDF header time-stamp.

The CIDF message layer shall insert a new sequence number. The sequence number is initialize to 0, and incremented for each message sent by the CIDF message layer.

The CIDF message layer shall compute the total message length and insert that length into the Length field.

The CIDF message layer should compute and insert the checksum prior to message transmission. The checksum is inserted prior to applying CIDF privacy or authentication mechanisms.

If CIDF privacy or authentication is being used, the CIDF message layer shall encrypt and generate the authentication data for the message based on the current security association in use with the recipient. If CIDF privacy or authentication is being used and no security association exists, then the message transmission request should be rejected.

4.2.2.2: Inbound Message Processing

If the Version field is not a valid CIDF version number (currently 1),

the CIDF message layer shall discard the message.

If CIDF privacy or authentication is being used, the CIDF message layer shall decrypt and authenticate the message, and discard the message on failure. On failure, due to lack of a valid security association, the CIDF message layer should send a response to the source. The response is the CIDF message layer header, with the Control Byte set to 4.

If the Checksum field is not 0, the CIDF message layer shall compute the message checksum (using the method described in RFC 793 and discard the message if the Checksum check fails.

If the Time Stamp field indicates an unexpected delay, the CIDF message layer should notify the application.

If the Destination Address is not the local CIDF node (i.e., the destination does not match the local node's address or any multicast address that the local node is using), the CIDF message layer shall determine the next CIDF hop (using the source route, if provided) and forward the message after adjusting the Sequence Number and Time Stamp. If the message includes a record route option, then the CIDF message layer shall enter its outgoing IP address if there is sufficient room in the record route structure and increment the route index. After processing, the CIDF node should compute the checksum as specified in RFC 793, and place the checksum in the Checksum field. Finally, the message layer shall apply the privacy and authentication transforms for the next CIDF hop and transmit the message.

4.2.3: Reliable Transmission Procedures

4.2.3.1: Outbound Message Processing

For reliable message transmission, the CIDF message layer shall maintain the round-trip latency and mean deviation values for each node with which the local component communicates. These values are used in determining the timeout values for message transmission. The CIDF message layer shall use the standard TCP algorithm for computing message layer timeouts.

On reliable message transmission, the CIDF message layer shall retain a copy of the message for retransmission purposes and set a timer for the message. If the timer expires before the message is acknowledged, the message layer shall retransmit the message up to an maximum of 5 retries.

On reception of an acknowledgement for the CIDF message, the CIDF message layer shall remove the message from the retransmission queue.

4.2.3.2: Inbound Message Processing

On message reception, the CIDF message layer shall send a CIDF acknowledgement to the source. If the message layer can deliver the message to the application layer, then the Control Byte shall be 1. Otherwise, the Control Byte shall be 2. The acknowledgement message is identical to the original message header except for the Control Byte.

The CIDF message layer shall use the source IP address and the sequence number to ensure that duplicate messages are not delivered to the application layer.

4.2.4: CIDF Privacy Procedures

4.2.4.1: Outbound Message Processing

The CIDF message layer encapsulates the CIDF Application Data with an CIDF privacy header and Trailer, encrypts the CIDF Application Data and CIDF privacy trailer. The original CIDF Header is retained, except the CIDF Next Type, which is modified to indicate that this is an CIDF encrypted message.

On message transmission, if the CIDF message layer is applying CIDF privacy mechanisms for the message, the CIDF message layer shall determine the security association (which determines the algorithm) for the message target, add any required padding, compute and insert the padding length in the trailer, insert the next header in the trailer, perform the cryptographic transform over the resulting plain-text message, and shall insert the security association identity (Key Generator Identity and SPI) before the resulting ciphertext. If an initialization vector is required for the cryptographic transform, it shall be inserted between the resulting ciphertext and the privacy header.

The next header in the CIDF message layer is then set to 50.

4.2.4.1.1 Message Encryption

The CIDF message layer encapsulates the original CIDF application data into the CIDF Application Data field, that includes any necessary padding, and encrypts the result (Application Data, Padding, Pad Length, and Next Header) using the Message Encryption Key, encryption algorithm, and algorithm mode indicated by the security association.

4.2.4.1.2 Encryption Algorithms

The security association specifies the encryption algorithm to be used. The CIDF privacy option is designed for use with symmetric encryption algorithms. Because the messages may arrive out of order, each message must carry any data required to allow the receiver to establish

cryptographic synchronization for decryption. This data may be carried explicitly in the Application Data field (e.g., as an IV as described above) or the data may be derived from the message header. Since the CIDF privacy option makes provision for padding of the plain-text, encryption algorithms employed with the CIDF privacy option may exhibit either block or stream mode characteristics.

4.2.4.2 Inbound Message Processing

Upon receipt of an CIDF message containing an CIDF privacy header, the CIDF message layer looks up the security association, and regenerates the CIDF application data.

4.2.4.2.1 Security Association Lookup

The Security Association information is included in the CIDF privacy header.

The CIDF message layer looks up the appropriate algorithm and Message Encryption Key for decryption, based on the SPI and Key Generator's identity from the CIDF privacy header.

If no valid algorithm and key exists for this message, the receiver MUST discard the message.

4.2.4.2.2 Message Decryption

The receiver decrypts the CIDF Application Data, Padding, Pad Length, and Next Header using the neighborhood Message Encryption Key that has been established for this neighborhood traffic. If an explicit IV is present in the payload field, it is input to the decryption algorithm per the algorithm specification. If an implicit IV is employed, a local version of the IV is constructed and input to the decryption algorithm per the algorithm specification.

After decryption, the original CIDF message is reconstructed and processed per the normal CIDF protocol specification. At a minimum, the Next Header field in the CIDF privacy trailer should be moved to the Next Header field in the CIDF header.

Note that there are two ways in which the decryption can "fail". The selected security association may not be correct or the encrypted CIDF message could be corrupted. (The latter case would be detected if authentication is selected for the security association, as would tampering with the SPI.)

4.2.5: CIDF Authentication Procedures

4.2.5.1 Outbound Message Processing

On message transmission, if the CIDF message layer is applying CIDF authentication mechanisms for the message, the CIDF message layer shall determine the security association (which determines the algorithm) for the message target, insert the length of the authentication header, insert the next header in the authentication header, shall insert the security association identity (Key Generator Identity and SPI) before the resulting ciphertext, and perform the cryptographic transform over the resulting message.

The next header in the CIDF message layer is then set to 51.

4.2.5.1.1 Integrity Check Value Calculation

The transmitter computes the Integrity Check Value (ICV) over the entire message using the ICV key, hashing algorithm, and algorithm mode indicated by the security association. Since the Authentication Data is not protected by encryption, a keyed authentication algorithm must be employed to compute the ICV.

If privacy is selected in conjunction with CIDF authentication, encryption is performed first, before the authentication. The encryption does not encompass the Authentication Data field. This order of processing facilitates rapid detection and rejection of replayed or bogus messages by the receiver, prior to decrypting the message, hence potentially reducing the impact of denial of service attacks. It also allows for the possibility of parallel processing of messages at the receiver (i.e., decryption can take place in parallel) with authentication.

4.2.5.1.2 Padding

No padding is required if the default 96-bit truncated Hashed Message Authentication Codes (HMAC) algorithm is used. However, if another authentication algorithm is used, padding MAY be required.

If an authentication algorithm creates an ICV with length less than an integral multiple of 32 bits, padding may be appended to the Authentication Data field to ensure a 32-bit multiple AH. Alternatively, the ICV may be truncated to a 32-bit multiple length.

In addition, if the authentication algorithm requires a multiple of a block size and the CIDF message with CIDF authentication header does not meet the block size requirements, zero-valued padding MUST be appended to the end of the CIDF message prior to ICV computation. This padding is not transmitted with the CIDF message.

4.2.5.1.3 Authentication Algorithms

The security association specifies the authentication algorithm used for the ICV computation. At the time of writing, one mandatory-to-implement algorithm and mode has been defined for CIDF authentication header. It

is based on the Hashed Message Authentication Codes using a SHA-1 hash value. The output of the HMAC computation is truncated to the leftmost **96 bits.**

4.2.5.2 Inbound Message Processing

Upon receipt of an CIDF message containing an CIDF authentication header, the CIDF message layer looks up the Security Association and verifies the Integrity Check Value.

4.2.5.2.1 Security Association Lookup

The Security Association information is included in the CIDF authentication header. The CIDF message layer looks up the appropriate algorithm and key for ICV computation, based on the SPI and Key Generator's identity from the CIDF authentication header.

If no valid algorithm and key exists for this message, the receiver MUST discard the message.

4.2.5.2.2 Integrity Check Value Verification

The receiver computes the ICV of the entire CIDF message using the specified authentication algorithm and the security association ICV key that has been established for this security association. If the computed ICV matches the ICV included in the Authentication Data field of the message, the CIDF message is valid and accepted. If the values do not match, the CIDF message layer MUST discard the CIDF message as invalid.

To validate the CIDF message, the CIDF message layer saves the ICV value in the CIDF authentication header and replaces it with zeros. Then the CIDF message layer performs the ICV computation over the entire message and compares the saved ICV value with the computed ICV value.

==========
==========
=
= APPENDIX A: Minimum CIDF Primitive Types Definition
=
==========
==========

The primitive types list enumerates the ways in which various data fields shall be encoded. The intent is to keep the primitive type list relatively small. Complex record structures should be defined within

the payload S-expression, while the definition of enumerated field arrays are left to SIDs.

```
#########################################################################
# Editor's Comment:  Please forgive the reliance on C terminology.
# These types should be applicable across programming languages.
#########################################################################
```

INTEGRAL TYPES: These are the core type primitives. These are the preferred data types for SIDs with associated fields that are processed or character encoding.

```
  char:         (7-bit ISO Invariant Code Set)
  char8:        (8-bit ascii, no idea what standard to suggest)
  byte:         (pure undefined octet, no sign relation)
  short:        (16-bit, signed)
  ushort:       (16-bit, unsigned
  long:         (32-bit, signed)
  ulong:        (32-bit, unsigned)
  float:        (floating point 32-bit, point to IEEE standard)
  double:       (floating point 64-bit, point to IEEE standard)
```

AGGREGATE TYPES: These are for SIDs that require arrays or structures.

An array is a collection of elements described by the same semantic identifier or type. An array can be of any length.

```
Example: (array byte) An array of bytes.
  (array record)  An array of records.
```

An arrayn is similar to array, but has a fixed number of elements.

```
Example: (arrayn 4 byte) An array consisting of 4 bytes.
  (arrayn 256 char8)  An array of 256 characters.
```

A structure is a sequence of possibly named elements of the specified types.

```
Example: (structure
   ((arrayn 256 char8) name)
   (ulong ref-count)
   (time create-time))
  A sequence consisting of a name, an array of 256 characters,
          a ref-count, an integer, and a create-time, a time.
```


Following are some predefined aggregate types:

```
string: (array char)
string8: (array char8)
```

```
ip4_address: (arrayn 4 byte)
ip6_address: (array 8 ushort)
header: (structure
                (byte version_id)
                (ulong msg_length)
                (timestamp msg_timestamp)
                (ulong sequence_num)
                (byte priority)
                (module_id source_ID)
                (module_id consumer_ID))

message: (structure
                (header msg_header)
                (payload msg_payload))

revision: (structure
                 (byte major)
                 (byte minor))

    [ Note: A generic time-stamp structure: NTP/UTC timestamp:
      integral part: time in seconds since 00:00:00 GMT,
      January 1, 1900 (proposed: if the high-bit is off,
      that it be relative to 06:28:16, February 7, 2036).
      fractional part: 32 bits (i.e., MSB = 1/2 sec, etc.)
      resolution of about 233 picoseconds (1 picosecond =
      10e-12 seconds). TBD: Accuracy estimation info.
      two components: accuracy of clock, estimate of drift. ]

timestamp: (structure
                  (ulong seconds)
                  (ulong fracsec))
```

SCALAR TYPES: There are no enumerated types in this specification, as there are in C, Ada, and Pascal. Rather, enumerated types shall be simple bitfield type structures, where SIDs build their own interpretation of the available bit patterns. This could save some tremendous headaches. Note, however, for portability it may be better for SIDs to simply utilize numeric values rather than bit manipulation when defining enumerated structures.

= Appendix B

B.1. Introduction

In the following sections, the SIDs defined for use in CIDF are described. GIDO producers SHOULD use these SIDs whenever an appropriate one is defined. If a SID has any applicable extensions, these are given

below the extended SID, indented. Extensions of extensions are further indented.

B.2. Verb SID Descriptions

Following each verb SID is a list of recommended role SIDs. This means that the sentence SHOULD contain role SIDs giving the described information (or must refer to an earlier instance which contains this information).

B.2.1. Motion Verb SIDs

Copy
Description: An object (or set of objects) was copied from one place to another. To be distinguished from Store by the fact that the Operand of Copy represents the *identifier* of the object being copied, and the Operand of Store represents the *value* being stored. Hence, if one wishes to express a string that is stored in a file, then that string is being Stored, and not Copied (and the appropriate verb SID should be used).
Recommended role SIDs:
Initiator: The entity responsible for initiating the copy.
Operand: The object being copied.
From: The original location of the object.
To: The new location of the copied object.

Move
Description: An object (or set of objects) was *moved* from one place to another. This is used when the object is persistent.
Compare Transmit.
Recommended role SIDs:
Initiator: The entity responsible for initiating the move.
Operand: The object being moved.
From: The original location of the object.
To: The new location of the moved object.

Store
Description: An object (or set of objects) was stored. To be distinguished from Copy by the fact that the Operand of Store represents the *value* being stored (and values do not have a source), and the operand of Copy represents the *identifier* of the object being copied.
Recommended role SIDs:
Initiator: The entity responsible for initiating the store.
Operand: The object being stored.
To: The new location of the stored object.

Remove
    Description: An object (or set of objects) was removed.
    Recommended role SIDs:
        Initiator: The entity responsible for initiating the remove.
        Operand: The object being removed.
        From: The original location of the object.

B.2.2. Process Verb SIDs

Execute
    Description: A program or command was executed.
    Recommended role SIDs:
        Initiator: The entity responsible for executing the program.
        Operand: The program being executed.

Interrupt
    Description: A program in execution was interrupted.
    Recommended role SIDs:
        Initiator: The entity responsible for interrupting the
        program.
        Operand: The program in execution that was interrupted.

Resume
    Description: An interrupted program was resumed.
    Recommended role SIDs:
        Initiator: The entity responsible for resuming the program.
        Operand: The interrupted program that was resumed.

Terminate
    Description: A program (either in execution or interrupted) was
    terminated.
    Recommended role SIDs:
        Initiator: The entity responsible for terminating the
        program.
        Operand: The program that was terminated.

B.2.3. Privilege Verb SIDs

AcquirePrivilege
    Description: An entity acquired a privilege.
    Recommended role SIDs:
        Initiator: The entity acquiring the privilege.
        Operand: The privilege in question.

LosePrivilege
    Description: An entity lost a privilege.
    Recommended role SIDs:
        Initiator: The entity losing the privilege.
        Operand: The privilege in question.

ChangeAttribute
    Description: An entity acquired a privilege.
    Recommended role SIDs:
        Initiator: The entity changing the attribute.
        Operand: The attribute in question.

HasAuthorizations
    Description: An entity has permission to do certain things
    to an object.
    Recommended role SIDs:
        Initiator: The entity operating on the object.
        Operand: The object being operated on.
        Authorizations: The permissions associated with this
        Initiator and Operand. Currently, this should use the
        Permissions SID (see Section B.5.7).

B.2.4. Transaction Verb SIDs

Request
    Description: An entity requested a second entity to perform an
    action.
    Recommended role SIDs:
        Initiator: The entity making the request.
        To: The entity receiving the request.
        Operand: A second-level sentence that contains the requested
        action. The Initiator of this second-level sentence MAY be
        the To of the request. (It may be the case that
        delegation is required.)

Login
    Description: An entity logs in (or attempts to do so) to a
    host.
    Recommended role SIDs:
        Initiator: The entity logging into the host.
        To: The host being logged into (ulch).

B.2.5. Communication Verb SIDs

BeginSession
    Description: A communication session is established between
    two entities.
    Recommended role SIDs:
        Initiator: The entity initiating the connection (the
        "active" mode side).
        To: The entity accepting the connection (the
        "passive" mode side).
        Operand: The connection and its associated attributes.
        (This might include a ReferAs clause, for instance.)

EndSession
Description: A communication session between two entities is ended.
Recommended role SIDs:
Initiator: The entity who had originally initiated the connection.
To: The entity who had originally accepted the connection.
Operand: The connection being closed. (Typically, this would consist only of a ReferTo clause.)

Transmit
Description: An entity sent an object to a second entity. This is used when the object is transient, as with a packet. Compare Move.
Recommended role SIDs:
Initiator: The entity responsible for starting the transmission. Typically the same entity as (and hence a possible referent of) the From.
From: The entity sending the object.
To: The entity receiving the object.
Operand: The object being sent.
Connection: Information identifying the connection, if applicable.

Block
Description: A transmission was blocked.
Recommended role SIDs:
Initiator: The entity blocking the transmission.
From: The entity sending the object.
To: The entity receiving the object.
Operand: The object being sent.
Connection: Information identifying the connection, if applicable.

B.2.6. Monitoring Verb SIDs

Snapshot
Description: A one-time observation of conditions.
Recommended role SIDs:
Observer: The entity observing the system.
PresentState: The conditions observed by the Observer.

ChangeState
Description: Conditions changed.
Recommended role SIDs:
Observer: The entity observing the system.
OldState: The old state of the system.
NewState: The new state of the system.

Filter
Description: A filter was matched by summarized occurences.
Recommended role SIDs:
Observer: The entity filtering the observations.
Filter: The filter against which to match the observations.
FilterStats: The statistics of observations matching the Filter.

B.2.7. Policy Verb SIDs

Require
Description: An entity required a second entity to perform an action.
Recommended role SIDs:
Initiator: The entity making the requirement.
To: The entity imposed by the requirement.
Operand: A second-level sentence describing the required action. See Request.

Recommend
Description: An entity recommended a second entity to perform an action.
Recommended role SIDs:
Initiator: The entity making the recommendation
To: The entity receiving the recommendation
Operand: A second-level sentence describing the recommended action. See Request.

Allow
Description: An entity allowed a second entity to perform an action.
Recommended role SIDs:
Initiator: The entity making the allowance.
To: The entity permitted to perform the action.
Operand: A second-level sentence describing the allowed action. See Request.

Forbid
Description: An entity forbade a second entity from performing an action.
Recommended role SIDs:
Initiator: The entity making the proscription.
To: The entity forbidden to perform the action.
Operand: A second-level sentence describing the forbidden action. See Request.

B.2.8. Subjunctive Verb SIDs

Do
Description: This is a request for action.
Recommended role SIDs:
Operand: A second-level sentence describing what is to be done. This sentence may lack an explicitly named Initiator, indicating that the recipient of the request is to ensure that the action is carried out.
Recommender: The entity recommending the action.

Diagnose
Description: This is a diagnosis.
Recommended role SIDs:
Operand: A second-level sentence indicating what the diagnosis is.
Analyzer: The entity making the diagnosis.

Predict
Description: This is a prognosis.
Recommended role SIDs:
Operand: A second-level sentence indicating what is predicted.
Analyzer: The entity making the prediction.

B.3. Role SID Descriptions

The below are SIDs that denote a *role*. That is, they are used to tag a group of attributes that together describe something that plays a role in a sentence. They are intended to be used somewhat generically, so that the same SID (e.g., Initiator) can be used with many different actions. Thus, their definitions here are generic as well. When used with specific verbs, they have specific interpretations; these are given with the definition of the verbs.

B.3.1. Operation Role SIDs

Initiator
Description: The entity responsible for causing something to happen.

Operand
Description: The object which is operated on. Often used in conjunction with Initiator, in which case it is the object that things happen *to*.

Authorizations
Description: The authorized activities permitted to an entity on an object.

Using
Description: The means used by the Initiator to perform some action. This is ordinarily a *program* or script which is executed by Initiator, which distinguishes it from the ByMeansOf construct (see Section B.4), which indicates that a *sentence* occurred as a way of having a second sentence occur. Therefore, a user might login "Using" a telnet program, but that same user would overload a system "ByMeansOf" causing that system to fail.

B.3.2. Time and Location Role SIDs

Before
Description: A time before which the conditions described by the sentence are observed.

After
Description: A time after which the conditions described by the sentence are observed.

While
Description: The time during which the conditions described by the sentence are observed.

AtTime
Description: The time at which the conditions described by the sentence are observed.

AtLocation
Description: The location at which the conditions described by the sentence are observed to have happened.

B.3.3. Motion Role SIDs

From
Description: The place from which movement occurs. This may also be used to identify an entity (i.e., active agent) sending an object.

To
Description: The place to which movement occurs. This may also be used to identify an entity receiving an object.

Through
Description: A place through which movement passes. This may also be used to identify an entity relaying an object. There may be more than one Through for a given verb. If there is an order to the Throughs, the GIDO producer SHOULD put them in chronological order. (To make this explicit, the producer MAY include an Epoch SID (see Section B.5.1) in this role.)

Connection
Description: A connection between two entities.

B.3.4. Filter Role SIDs

Filter
Description: The template against which a component is matching observations, typically for statistics.

FilterStats
Description: The statistics on which observations match the associated Filter.

B.3.5. Object Attribute Role SIDs

Owner
Description: The entity with ownership rights to the object. If specific assignment of ownership is lacking, then this indicates that the entity has the right to control access to the object.

Host
Description: The host on which the object resides.

Certifier
Description: The entity which vouches for the identity of the object. When applied to a certificate, it represents the CA for that certificate; when applied to a Kerberos principal, it represents the KDC; when applied to a hostname, it represents a (possibly secure) DNS server.

Parent
Description: The entity which created the object. When applied to a Unix process, it represents the parent of the process.

B.3.6. Status Role SIDs

Outcome
Description: The outcome of an action which was *attempted*.

Context
Description: Additional information regarding an event. This is where the duration of an event, for instance, would go.

B.3.7. Meta-Role SIDs

Observer

Description: The entity observing the conditions described by the sentence.

Analyzer
Description: The entity performing the analysis described by the sentence.

Recommender
Description: The entity prescribing the action described by the sentence.

B.4. Conjunction SIDs

CausallyRelated
Format: (CausallyRelated <Sentence1> <Sentence2>)
Description: Sentence1 and Sentence2 both occurred, and are causally related. That is, either one helped cause the other, or a third sentence helped cause both.

HelpedCause
Format: (HelpedCause <Sentence1> <Sentence2>)
Description: Sentence1 and Sentence2 both occured, and Sentence1 was a contributing factor to Sentence2. Sentence2 must denote an event.

IntentionallyHelpedCause
Format: (IntentionallyHelpedCause <Sentence1> <Sentence2>)
Description: Sentence1 and Sentence2 both occured, and Sentence1 was, and was intended to be, a contributing factor to Sentence2. Sentence2 must denote an event.

CommonCause
Format: (CommonCause <Sentence1> <Sentence2> ... <SentenceN>)
Description: Sentence1 through SentenceN all occurred, and all have a common cause.

ByMeansOf
Format: (ByMeansOf <Sentence1> <Sentence2> ... <SentenceN>)
Description: Sentence1 through SentenceN all denote events that happened. Some intentional agent (person or program) accomplished event 1 by means of accomplishing event 2, accomplished event 2 by means of accomplishing event 3, and so on.

InOrder
Format: (InOrder <Sentence1> <Sentence2> ... <SentenceN>)
Description: Sentence1 through SentenceN all denote events that occurred. The events occurred in the order given. ending(event 1) <= beginning(event 2),

ending(event 2) <= beginning(event 3), ...

B.5. Atom SIDs

B.5.1. Base SIDs

Epoch
Type: timestamp
Description: The moment at which something occurred.

Duration
Type: float
Description: The duration of an occurrence, in seconds.

Size
Type: ulong
Description: The length of an object, in bytes.

Certainty
Type: float
Description: The certainty of an observation, expressed as the observer's estimate of the probability that the entire observation is true.

Severity
Type: byte
Description: The severity of an event, as estimated by the observer, with more severe events being given a higher number. Zero (0) represents the minimum severity, and 255 represents the maximum severity. [Editor's Note: We solicit discussion on guidelines for determining intermediate levels of severity.]

ReturnCode
Type: byte
Description: This is an enumerated value. The constraints on this are that zero (0) MUST mean success; failure is any non-zero value.

ReturnCode (ExtendedBy CIDFReturnCode)
Type: byte
Description: This is an enumerated value with greater semantics than a generic return code:
0 = success
1 = failed
2 = pending
3-255 = reserved

OSName

```
        Type: string
        Description: The name of an operating system.  (For instance,
        "SunOS").

    ObservationSourceType
        Type: string
        Description: A name describing the source of data used to
        generate an observation.
```



```
    ObjectType
        Type: byte
        Description: This is an enumerated value.
            0     = reserved
            1     = file
            2     = file_system
            3     = memory
            4     = CPU_time
            5     = peripheral
            6     = URL
            7     = network_packet
            8     = program
            9-255 = reserved

    ObjectName
        Type: string
        Description: The name of an object.

        (ExtendedBy FileSystemName)
            Type: string
            Description: A file system name, given in
            "hostname:directory-pathname" format, if object type
            is file system.

        (ExtendedBy DeviceName)
            Type: string
            Description: A name for the device, if object type is
            peripheral.

        (ExtendedBy URL)
            Type: string
            Description: A URL to find the object, given in
            "scheme:locator" format, if object type is URL.

    ObjectCreated
        Time: timestamp
        Description: Depending on applicability, the time at which
        the object was (last) created.

    ObjectModified
```

Time: timestamp
Description: Depending on applicability, the time at which the object was last modified.

ObjectAccessed
Time: timestamp
Description: Depending on applicability, the time at which the object was last accessed.

ProgramName
Type: string
Description: The name of a program, as distinguished from its filename. If a program is moved from one place in a directory to another, its filename may change, but its program name stays the same. An example of a program name is "PowerPoint".

DeveloperName
Type: string
Description: The name of the entity responsible for developing a program. An example is "Microsoft".

VersionNumber
Type: string
Description: The version number of an application. An example is "5.0".

Comment
Type: string
Description: An annotation.

B.5.2. Connector

ReferAs
Type: string
Description: An identifier to be used later as a referent. (See Section 2.4.)

ReferTo
Type: string
Description: An identifier referring to an earlier referent. (See Section 2.4.)

B.5.3. Process SIDs

Priority
Type: short
Description: The priority of a process, with high-priority processes being given a lower number.

RevPriority
    Type: short
    Description: The priority of a process, with high-priority
    processes being given a larger number.

ProcessName
    Type: short
    Description: The process's name (typically used for daemon or
    service name).

ProcessID
    Type: ushort
    Description: The process's ID.

ProcessID (ExtendedBy SessionID)
    Type: ushort
    Description: Denotes an ID relating to a process that carries
    a session.

ProcessStatus
    Type: byte
    Description: This is an enumerated value.
        0   = active   /* running */
        1   = suspended   /* awaiting an OS action */
        2   = killed   /* terminated by external signal */
        3   = finished   /* terminated internally */
        4   = unknown (no such process?)
        5-255 = reserved

SystemTime
    Type: float
    Description: Time spent (by a process) in the system/kernel,
    in seconds.

UserTime
    Type: float
    Description: Time spent (by a process) in user space, in
    seconds.

B.5.4. User SIDs

EMailAddress
    Type: string
    Description: The e-mail address of an entity.

RealName
    Type: string
    Description: The given name, as known, of an entity.

PrincipalName
    Type: string
    Description: A persistent name associated with a user.

UserName
    Type: string
    Description: The name associated with a user account.

UserID
    Type: ushort
    Description: The user ID number for a user.

PersistentUserName
    Type: string
    Description: A user name associated with a login session.

PersistentUserID
    Type: ushort
    Description: The user account ID.

CurrentUserName
    Type: string
    Description: A user name associated with the current interactive
    session.

CurrentUserID
    Type: ushort
    Description: The current user ID.

EffectiveUserName
    Type: string
    Description: A user name associated with a process.

EffectiveUserID
    Type: ushort
    Description: The effective user ID.

GroupName
    Type: string
    Description: The name associated with a user group. This can
    be associated with either a user or an object such as a file.

EffectiveGroupName
    Type: string
    Description: The name associated with the group of an effective
    user.

GroupID
    Type: ushort

Description: A user group ID. This can be associated with
either a user or an object such as a file.

GroupUUID
Type: 16-byte array
Description: A UUID associated with a group. This has stronger
guarantees of non-duplication than GroupID.

B.5.5. Distributed/Networking SIDs

HostName
Type: string
Description: The hostname associated with a host.

(ExtendedBy FQHostName)
Type: string
Description: The fully qualified hostname.

ServerDNSName
Type: string
Description: The name of a server associated with a user
or an object. This is always a fully qualified DNS name.

DomainName
Type: string
Description: An administrative domain name.

(ExtendedBy FQDomainName)
Type: string
Description: The fully qualified domain name.

DomainID
Type: ulong
Description: An administrative domain's ID.

DomainUUID
Type: 16-byte array
Description: A UUID associated with an administrative domain.

DataLinkProtocol
Type: byte
Description: Enumerated value.
0 = reserved
1 = Ethernet
2 = Token ring
3 = ARC net
4 = IEEE 802.5, SNAP header
5 = IEEE 802.2, FDDI
6 = IEEE 802.3, MAN

```
            7     = SLIP
            8     = PPP
            9-255 = reserved

    NetworkProtocol
        Type: byte
        Description: Enumerated value.
            0     = reserved
            1     = IPv4
            2     = IPv6
            3     = ICMP
            4     = ARP
            5     = RARP
            6-255 = reserved

    TransportProtocol
        Type: byte
        Description: Enumerated value.
            0     = reserved
            1     = TCP
            2     = UDP
            3-255 = reserved
```

B.5.5.1. Data Link Layer Attribute SIDs

```
    EthernetAddress
        Type: 6-byte array
        Description: The ethernet address associated with a host.

    EtherPreamble
        Type: 8-byte array
        Description: The ethernet preamble.

    EtherType
        Type: ushort
        Description: The ethernet type.
```



```
    EtherFrameCheckSeq
        Type: ulong
        Description: The ethernet frame check sequence number.
```

B.5.5.2. Network Layer Attribute SIDs

```
    IPV4Address
        Type: ulong
        Description: The IPv4 address associated with a host.

    IPV4Mask
        Type: ulong
```

Description: An IPv4 network mask.

IPV4Port
Type: ushort
Description: An IPv4 port.

IPV4verIHL
Type: byte
Description: 8 bit representation of IP header version ID (bits 0-3) and header length (bits 4-7).

IPV4servicetype
Type: byte
Description: The IPv4 Type of Service (TOS). Only bits 3-6 are significant. They are, in order, Minimize Delay, Maximize Throughput, Maximize Reliability, and Minimize Monetary Cost.

IPV4totallength
Type: ushort
Description: The IPv4 total packet length. This is the length of the entire packet, header included. Subtracting the header length (determined from IPV4verIHL) gives the length of the data portion.

IPV4identifier
Type: ushort
Description: The IPv4 packet identifier. Typically sequential.

IPV4flags
Type: byte
Description: The IPv4 flags field. Only bits 1 and 2 are significant. They are, in order, Don't Fragment and More Fragments.

IPV4fragoffset
Type: ushort
Description: The IPv4 offset of a datagram fragment.

IPV4ttl
Type: byte
Description: The IPv4 packet time-to-live (maximum number of routers to relay the packet).

IPV4protocol
Type: byte
Description: The IPv4 protocol number. This is an enumerated value. A full table of the enumeration can be found in RFC 1700. [There has to be something more recent than that.]

IPV4checksum
Type: ushort
Description: The IPv4 header checksum. It does not cover any of the data portion.

B.5.5.3. Transport Layer SIDs

TCPPort
Type: ushort
Description: A TCP port number.

TCPsequencenumber
Type: ulong
Description: The TCP sequence number. Identifies the first byte of the stream in this packet. The first byte of the stream as a whole is indexed number zero.

TCPacknumber
Type: ulong
Description: The TCP ack number. Identifies the first byte of the stream in the next (expected) packet.

TCPwindow
Type: ushort
Description: The TCP window size, in bytes.

TCPchecksum
Type: ushort
Description: The TCP packet checksum. This covers both the header *and* the data portions of the packet.

TCPurgentpointer
Type: ushort
Description: The TCP urgent pointer. Presence of this SID presumes that the URGent bit was set. Number of bytes to get to the last byte of urgent data, starting from the first byte of this packet, inclusive.

TCPMSS
Type: ushort
Description: The TCP maximum segment size.

TCPflags
Type: byte
Description: Bit 0 = URG, Bit 1 = ACK, Bit 2 = PSH, Bit 3 = RST, Bit 4 = SYN, Bit 5 = FIN, Bits 6-7 = undefined.

TCPflagsmask
Type: byte

Description: A bitmask over the TCPflags field.

UDPPort
Type: ushort
Description: A UDP port number.

UDPlength
Type: ushort
Description: The UDP packet length. Covers both the header and the data.

UDPchecksum
Type: ushort
Description: The UDP packet checksum. This covers both the header *and* the data.

B.5.6. Statistics SIDs

SampleCount
Type: ulong
Description: The number of samples that were analyzed.

MatchCount
Type: ulong
Description: The number of samples that matched the filter.

DistinctMatchCount
Type: ulong
Description: The number of non-duplicated samples that matched the filter.

MismatchCount
Type: ulong
Description: The number of samples that did not match the filter.

DistinctMismatchCount
Type: ulong
Description: The number of non-duplicated samples that did not match the filter.

Anomalousness
Type: float
Description: This represents the degree to which the statistics are unexpected, represented as a probability that a random sample set of the given size would exhibit the given behavior *or more extreme*. This can only be used with filter match or mismatch counts. With match counts, this represents the estimated probability of finding at least that many matches; with mismatch counts, the estimated probability of finding at least that many mismatches.

Deviation
Type: float
Description: The distance of the statistics from the mean value, expressed as multiples of the standard deviation. If the value is lower than expected, then the value of this field is negative; if the value is higher than expected, then the value of this field is positive.

B.5.7. Access Control SIDs

AccessControlList
Type: string
Description: An access control list, in some format. This ought to be standardized, but isn't.

MACLabel
Type: string
Description: The security label of the user.

MACEffectiveLabel
Type: string
Description: The security label of the effective user identifier.

MACClearances
Type: string
Description: The clearances of the user.

MACObjectLabel
Type: string
Description: The security label of an object.

Permissions
Type: byte
Description: This is a bitfield.
0 = read (MSB)
1 = append
2 = modify
3 = execute
4-7 = reserved

B.5.8. Uninterpreted SIDs

CharSID
Type: char
Description: A char type with no defined semantics.

ByteSID
Type: byte

Description: A byte type with no defined semantics.

ShortSID
Type: short
Description: A short type with no defined semantics.

UShortSID
Type: ushort
Description: A ushort type with no defined semantics.

LongSID
Type: long
Description: A long type with no defined semantics.

ULongSID
Type: ulong
Description: A ulong type with no defined semantics.

FloatSID
Type: float
Description: A float type with no defined semantics.

DoubleSID
Type: double
Description: A double type with no defined semantics.

StringSID
Type: string
Description: A string type with no defined semantics.

B.5.9. Packaged SIDs

The following SIDs are grouped into packages. Each package, generally speaking, covers a domain of interest. The intent is to group SIDs together that would be useul to any component interested in a specific area. This organization confers the following benefits.

* Components will be able to concisely specify which SIDs they do and do not understand, without being required to support a mandated set of SIDs.
* Components will not be required to support SIDs for which they have no use.
* Newly developed SIDs can be dropped into place with minimal fuss.

Some SIDs within packages extend earlier defined SIDs. As noted in the main text, this means that the interpretation of such SIDs builds upon the meaning of the extended SIDs.

B.5.9.1. Kerberos SIDs

This package contains SIDs related to Kerberos activity: Kerberos extensions (as enumerated by preauthentication field types), Kerberos logging, Kerberos error types, and identities.

KerberosV5PreauthType
Type: ushort
Description: The Kerberos V5 preauthentication field type.

KerberosV5EncType
Type: ushort
0 = NULL (no encryption)
1 = ENCTYPE_DES_CBC_CRC (DES in CBC mode with CRC32 checksum)
2 = ENCTYPE_DES_CBC_MD4 (DES in CBC mode with MD4 checksum)
3 = ENCTYPE_DES_CBC_MD5 (DES in CBC mode with MD5 checksum)
4 = ENCTYPE_DES_CBC_RAW (DES in CBC mode raw (no checksum))
5 = ENCTYPE_DES3_CBC_SHA (DES in CBC mode with SHA1 checksum)
6 = ENCTYPE_DES3_CBC_RAW (DES in CBC mode raw (no checksum))
7-510 = reserved
511 = unknown encryption type
Description: The Kerberos V5 encryption type used.

KerberosLogUserNumber
Type: string
Description: The user number associated with this audit log entr
be well-formed, it should be a six-digit number, although other
information may be inserted here, if that six-digit number is no
present.

KerberosLogEventType
Type: byte
1 = A ticket was requested by this user.
2 = A service (application) was requested by this user.
3 = The allowed threshold for initial ticket requests has been e
4 = User has been blacklisted by the KNSC (Kerberos system).
5 = User blacklist has been cleared by the KNSC.
6 = Denied request.
Description: An enumerated type indicating the type of event bei
logged.

KerberosLogEventStatus
Type: byte
For KerberosLogEventType = 1,
1 = Successful activity.
2 = Failed ticket request: Invalid computing level.
3 = Failed ticket request: Expired password.
4 = Failed ticket request: Access failure.

5 = Failed ticket request: Unknown user.
6 = Failed ticket request: Invalid level for source.
For KerberosLogEventType = 2,
1 = Successful activity.
2 = Failed service request: Invalid computing level.
3 = Failed service request: Unknown service.
4 = Failed service request: Expired ticket.
5 = Failed service request: Invalid level for destination.
Description: An enumerated type indicating, for various event ty
the current status of the event at the time of log entry.

KerberosLogEventLevel
Type: char
u = Unclassified.
p = PARD.
c = Confidential.
s = Secret.
Description: The user's requested computing level during
authentication.

KerberosLogEventService
Type: string
Description: Thirty-character name associated with the service
requested by the user. This may be an application (such as
telnet) or a node such as the CFS. A single '?' indicates that
the event service name was unknown.

ReturnCode (ExtendedBy KerberosV5Error)
Type: byte
Description: Indicates a Kerberos V5 error code. As
before, zero indicates success. The complete
enumeration may be found in RFC 1510.

PrincipalName (ExtendedBy KerberosName)
Type: string
Description: A Kerberos principal name, unparsed.
An example is "joe@WORK.COM". Its two extensions
ought to be self-explanatory.

(ExtendedBy KerberosV4Name)

(ExtendedBy KerberosV5Name)

ServerDNSName (ExtendedBy KerberosKDCName)
Type: string
Description: Indicates the host name of the Kerberos
server.

DomainName (ExtendedBy KerberosRealmName)

Type: string
Description: A Kerberos realm name.

B.5.9.2. DCE SIDs

This package contains SIDs related to DCE. It currently contains only identity-related SIDs, but it may also contain SIDs relating to logging, errors, and privileges.

GroupName (ExtendedBy DCEGroupName)
Type: string
Description: A DCE group name.

GroupUUID (ExtendedBy DCEGroupUUID)
Type: 16-byte array
Description: A DCE group UUID.

DomainName (ExtendedBy DCECellName)
Type: string
Description: A DCE cell name.

DomainUUID (ExtendedBy DCECellUUID)
Type: 16-byte array
Description: A DCE cell UUID.

B.5.9.3. AFS SIDs (For instance,

"SunOS"). (For instance,
"SunOS"). (For instance,
"SunOS"). (For instance,
"SunOS"). (For instance,
"SunOS").

This package contains SIDs related to DCE. It currently contains identity-related SIDs and ACL-related SIDs, but it may also contain SIDs relating to logging and errors.

AFSProtectionGroupName
Type: string
Description: The protection group associated with a given ACL.
This is a combination of the form <cell>:<group>:<user>.

AFSACL
Type: char
r = Read.
l = Lookup.
i = Insert.
d = Delete.
w = Write.
k = locK.

a = Administer.
Description: The access permission granted on the directory, to
a given AFSProtectionGroup.

GroupName (ExtendedBy AFSGroupName)
Type: string
Description: A AFS group name.

GroupUUID (ExtendedBy AFSGroupUUID)
Type: 16-byte array
Description: A AFS group UUID.

ServerDNSName (ExtendedBy AFSServerName)
Type: string
Description: Indicates the name of the host serving
the AFS file within the cell.

DomainName (ExtendedBy AFSCellName)
Type: string
Description: An AFS cell name.

DomainUUID (ExtendedBy AFSCellUUID)
Type: 16-byte array
Description: An AFS cell UUID.

B.5.9.4. CIDF Box Reporting SIDs

This package contains SIDs that convey information about the status and configuration of CIDF-compliant components (referred to as "boxes").

BoxRecsPerSec
Type: ulong
Description: The number of records per second received since upt

BoxBytesPerSec
Type: ulong
Description: The number of bytes per second received since uptim

BoxSentRecsCnt
Type: ulong
Description: The total number of records received since uptime.

BoxSentByteCnt
Type: ulong
Description: The total number of bytes received since uptime.

BoxErrorCnt
Type: ulong
Description: The total number of errors received since uptime.

```
    BoxWarnDesc
        Type: string
        Description: A narrative description of the warning.

    BoxEventStreamID
        Type: byte
                  0 = OS-audit
                  1 = packet-data
                    :
                    :
                 15 = reserved
                 16 = available
                    :
                255 = available
        Description: A byte value identifying the message stream type
```

B.5.9.5. Enumerated Action SIDs

This package contains enumerated lists of actions.

```
    FTPCommand
        Type: string
        Description: The name of an FTP command used during a session.
```



```
    ResourceAction
        Type: byte
        Description: An enumeration of all the actions that may be
        taken on a resource object.
            0 = access
            1 = chdir
            2 = chmod
            3 = chown
            4 = chroot
            5 = close
            6 = create
            7 = link
            8 = mount
            9 = open
            10 = read
            11 = umount
            12 = unlink
            13 = write
            14+ = reserved
```



```
    OSAction
```

```
Type: short
Description: This SID enumerates the possible OS audit/syslog ev
that may be seen in an audit trail.
    0 = void
    1 = discon
    2 = access
    3 = open
    4 = write
    5 = read
    6 = delete
    7 = create
    8 = rmdir
    9 = chmod
    10 = exec
    11 = chown
    12 = link
    13 = chdir
    14 = su
    15 = bad_su
    16 = exit
    17 = logout
    18 = uncat
    19 = rsh
    20 = passwd
    21 = rmount
    22 = passwd_auth
    23 = kill
    24 = core
    25 = ptrace
    26 = truncate
    27 = utimes
    28 = fork
    29 = chroot
    30 = mknod
    31 = halt
    32 = reboot
    33 = shutdown
    34 = boot
    35 = set_time
    36 = setuid
    37 = setgid
    38 = audit_config
    39 = is_promiscuous
    40 = connect
    41 = accept
    42 = bind
    43 = socketoption
    44+ = reserved
```

B.5.9.6. Unix SIDs

This package contains Unix-related SIDs: error codes and identities. It may also contain other Unix-specific SIDs.


```
    ReturnCode (ExtendedBy UnixErrno)
        Type: byte
        Description: Indicates a Unix errno value.  Thus,
        1 is EPERM, 2 is ENOENT, 3 is ESRCH, and so forth.
        This should be standard through 32 (EPIPE).  After
        that, the values are not standardized, so we may see
        this extension further ExtendedBy LinuxErrno, for
        example.  To be explicit about UnixErrno, we reproduce
        the enumeration below.  (Depending on the context,
        this may not need to be generated by an actual Unix
        process, but may also be used by (for example) A-boxes
        explaining why an action failed.)
             0     = SUCCESS  /* Inherited from ReturnCode */
             1     = EPERM    /* Operation not permitted */
             2     = ENOENT   /* No such file or directory */
             3     = ESRCH    /* No such process */
             4     = EINTR    /* Interrupted system call */
             5     = EIO      /* I/O error */
             6     = ENXIO    /* No such device or address */
             7     = E2BIG    /* Arg list too long */
             8     = ENOEXEC  /* Exec format error */
             9     = EBADF    /* Bad file number */
            10     = ECHILD   /* No child processes */
            11     = EAGAIN   /* Try again */
            12     = ENOMEM   /* Out of memory */
            13     = EACCES   /* Permission denied */
            14     = EFAULT   /* Bad address */
            15     = ENOTBLK  /* Block device required */
            16     = EBUSY    /* Device or resource busy */
            17     = EEXIST   /* File exists */
            18     = EXDEV    /* Cross-device link */
            19     = ENODEV   /* No such device */
            20     = ENOTDIR  /* Not a directory */
            21     = EISDIR   /* Is a directory */
            22     = EINVAL   /* Invalid argument */
            23     = ENFILE   /* File table overflow */
            24     = EMFILE   /* Too many open files */
            25     = ENOTTY   /* Not a typewriter */
            26     = ETXTBSY  /* Text file busy */
            27     = EFBIG    /* File too large */
            28     = ENOSPC   /* No space left on device */
            29     = ESPIPE   /* Illegal seek */
            30     = EROFS    /* Read-only file system */
            31     = EMLINK   /* Too many links */
            32     = EPIPE    /* Broken pipe */
```

```
        33-255 = undefined

    ObjectName (ExtendedBy UnixPathName)
        Type: string
        Description: A fully expanded Unix pathname, if object
        type is file.
```



```
    Priority (ExtendedBy UnixNiceness)
        Type: short
        Description: The process's Unix nice value.  User processes
        are restricted to positive (lower-priority) niceness values.

    ObjectName (ExtendedBy DeviceName) (ExtendedBy UnixFullDeviceName)
        Type: string
        Description: Indicates a full Unix device name, such
        as "/dev/ttyp0".  A simple "ttyp0" is not a valid
        value if this extension is present.

    UserName (ExtendedBy UnixUserName)
        Type: string
        Description: A Unix account name.

    UserID (ExtendedBy UnixUID)
        Type: ushort
        Description: A Unix UID.

    PersistentUserName (ExtendedBy UnixAUserName)
        Type: string
        Description: A Unix audit (real) user name.

    PersistentUserID (ExtendedBy UnixAUID)
        Type: ushort
        Description: A Unix audit (real) user ID.

    CurrentUserName (ExtendedBy UnixCUserName)
        Type: string
        Description: A Unix current user name.

    CurrentUserID (ExtendedBy UnixEUID)
        Type: ushort
        Description: A Unix current user ID.

    EffectiveUserName (ExtendedBy UnixEUserName)
        Type: string
        Description: A Unix effective user name.

    EffectiveUserID (ExtendedBy UnixEUID)
        Type: ushort
        Description: A Unix effective user ID.
```

GroupName (ExtendedBy UnixGroupName)
Type: string
Description: The name of a Unix group.

EffectiveGroupName (ExtendedBy UnixEGroupName)
Type: string
Description: The name of a Unix effective group.

UnixPermissions
Type: string
Description: The permissions associated with a Unix file, expressed in "04777" format. For example, "00544" means readable and executable by the owner, readable only by anyone else.

B.5.9.7. DOS SIDs

This package contains DOS-specific SIDs.

ObjectName (ExtendedBy DOSPathName)
Type: string
Description: A fully expanded DOS pathname, if object type is file.

B.5.9.8. X.500 SIDs

This package contains X.500 SIDs. It currently contains only identity-related SIDs, but it may also contain SIDs conveying information about certificates, directory servers, and algorithms.

RealName (ExtendedBy X500CommonName)
Type: string
Description: An X.500 Common Name.

PrincipalName (ExtendedBy X500DistName)
Type: string
Description: An X.500 Distinguished Name, encoded as in RFC 1779.

Expiration Date

This draft expires September 18th, 1998.

Authors

Stuart Staniford-Chen
Department of Computer Science,
UC Davis,

Davis, CA 95616
Phone:
(707) 825-0836
Fax:
(707) 826-7571
E-mail: stanifor@cs.ucdavis.edu

Brian Tung
USC Information Sciences Institute
4676 **Admiralty Way Suite 1001**
Marina del Rey CA 90292
Phone: (310) 822-1511 x135
Fax: (310) 823-6714
E-mail: brian@isi.edu

Phil Porras
SRI
Phone: (650) 859-3232
Fax: (650) 859-2844
E-mail: porras@csl.sri.com

Cliff Kahn
The Open Group Research Institute
11 **Cambridge Center**
Cambridge MA 02142
Phone: (617) 621-7221
Fax: (617) 621-8696
E-mail: c.kahn@opengroup.org

Dan Schnackenberg
Boeing Information, Space and Defense Systems
P.O. Box 3999
MS 88-12
Seattle WA 98124
Phone: (253) 773-8231
Fax: (253) 773-1015
E-mail: dan@baker.ds.boeing.com

Rich Feiertag
Trusted Information Systems
444 **Castro Street, Suite 800**
Phone: (415) 962-8885 x3012
Fax: (415) 962-9330
E-mail: feiertag@tis.com

Maureen Stillman
Odyssey Research Associates
33 **Thornwood Drive, Suite 500**

Phone: (607) 266-7123
Fax:
(607) 257-1972
E-mail: maureen@oracorp.com